UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND INFORMATICS
Institute of Computer Science

Timo Petmanson

Pattern based fact extraction from Estonian texts

Master's Thesis (30cp)

Supervisor: Sven Laur, D.Sc (Tech)

Author:.............................“.....“ May 2012
Supervisor: ....”.....“ May 2012
Admitted to thesis defense
Professor:............................”.....“ May 2012

Tartu 2012

Contents

Introduction

Throughout the history of mankind, the facts and knowledge have been transferred from generations to generations in various forms. In the beginning, it was mostly done orally and with time various different written forms of text appeared. Nowadays, we have reached the era where much of the information and knowledge is spread around digitally and held in internet. Popular sites such as *Wikipedia* are full of facts and knowledge easily accessible by everyone. Different news portals allow information to spread around the globe in minutes. Much of the data is stored either as entity-relationship databases, XML documents or other structured formats.

The need for automatically assessing this information has become a major goal in fields like *artificial intelligence (AI)* and *information retrieval* (IE). Being able to extract various relations from the texts help different applications to be more practical for the users. For example, a news recommender can make more precise suggestions, if it has additional information about the article that simpler methods cannot provide. Relations in medical information such as anamnesis can help doctors to better analyze and diagnose the patients. Using *natural language processing* (NLP) techniques, which involve deep linguistic analysis, we can make computers understand text to a certain degree. However, natural language processing is one of the most difficult problems since words and language constructions have often ambiguous meaning that cannot be resolved without extensive cultural background. Still, some facts are easier to deduce than the others.

Semantic web, sometimes also referred as Web 3.0 is a future goal lead by *World Wide Web Consortium*, with aim that the future web content should be mostly in structured form that could be directly processed by computers. *Resource Description Framework* (RDF) and *Web Ontology Language* (OWL) are steps in the direction. However, semantic web is complex and it has some rough corners. For example, there is no easy way to represent *n*-ary relations [HW06]. Currently, it is also not very widely adopted and possibly this is the situation in the future as few may choose to follow the standards. Free-form text is most popular resource of knowledge today and tomorrow.

In this work, we consider unary, binary and ternary relations between the words that can be deduced form a single sentence. For instance, sentence

President Obama meets President Ilves in Kadriorg.

and its Estonian translation

President Obama kohtub president Ilvesega Kadriorus.

expresses a ternary relation between words “Obama”, “Ilves” and “Kadriorg”. Intuitively, such relations can be captured with patterns. A *pattern* in this context is a sequence of tokens that can be matched against sentences. As we want to extract relations between words, some of these tokens must be labeled as entities to be extracted. For example, we might use the following pattern

X meets president Y in Z

for capturing the relation described above where X, Y and Z are placeholders for the words. We call these tokens as *extraction tokens*.

Description of relations in terms of patterns is not unique, as one can come up with many ways to describe patterns and good patterns might occasionally match words that are not related. Let $\mathcal{C}$ be a set of sentences (*corpus*) and r a relation we are interested in. For each pattern p we can talk about false positives and false negatives. A *false positive* is a match that creates relation between the words that is not supported by the sentence and *false negative* is a match that should have been made to extract the relation supported by the sentence. The corresponding rates of false positives $\mathsf{fp}(p)$ and false negatives $\mathsf{fn}(p)$ show how specific or sensitive the pattern is. For example, the pattern

X meets president Y in Z

is an instance of a rather specific pattern. It yields only matches from sentences with a very specific structure, thus it is less likely to give false positive matches. Another pattern

X ___ *president Y* ___ *Z*

is more sensitive and puts less constraints on the sentence structure (the blanks accept any word), thus lowering the false negative rate. But due to high diversity of Estonian language, it is highly unlikely that a single pattern can cover the entire relation. Hence, our main task is to find a list of patterns $\mathcal{P}$ such that we could extract most relations form the text with as few spurious relations as possible.

Although the methods and approaches we study in this work focus on processing the Estonian language, we write this work in English. There are several reasons: it is more natural to express IT-related technical terms in English; the reader may not understand Estonian; the methods could be used with modifications on other languages as well.

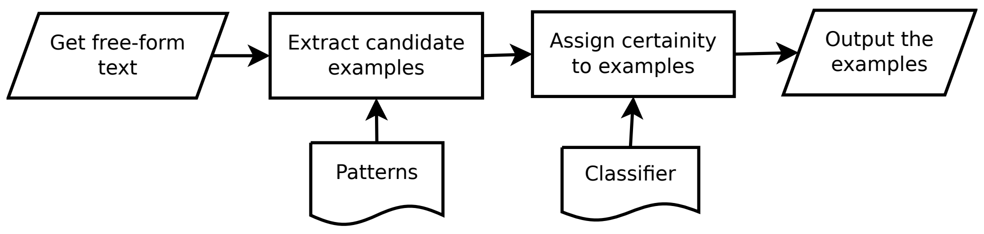


Figure 1: The process of fact extraction from input texts. Patterns match certain locations in the text and yield candidate examples, where classifier assigns a certain probability that the candidate pattern is true positive. The certainty can be used to configure to output a few, but very likely true positive examples or many examples, with greater likelihood of having false positives.

Problem statement. The goal of this work is to define and test a set of pattern abstraction methods, which working in tandem with a classifier can extract candidate examples from Estonian free-form texts as depicted in Figure 1. Most crucial part is how to find the best set of patterns, which describe the relation fairly well.

The reason for doing pattern based fact extraction in Estonian is that most of the work done in language technology related fields have emphasis on English. Estonian is a modern language applied in every-day use in communications, internet and government levels. The need for language technology support is crucial in nowadays Estonian information society to make the language sustainable and more easily usable in large-scale information systems.

We could also try to translate existing ontologies and knowledge bases constructed for English such as YAGO-NAGA [KRSW09] or DBpedia [ABK$^+$07], but still we cannot directly use the approaches used for fact extraction from free-texts due to various differences in the languages. Also, we might not be able to take advantage of current language technology tools for Estonian.

Achievable sensitivity and specificity is determined by the amount of utilized contextual information. Depending on the exact syntax and the number of pre-processing steps, patterns can encode various levels of syntactical, morphological and semantic information about words in the sentence. Hence, our ability to extract relations from sentences is greatly determined by the tokens which form patterns. As perfect description of relations is usually infeasible, we can consider two basic optimization tasks in our setting.

First, we can try to find a set of patterns $\mathcal{P}$ such that the rate of true positives (recall) $\mathsf{tp}(\mathcal{P})$ is maximal while the rate of false negatives $\mathsf{fn}(\mathcal{P})$ is small enough:

$$\begin{array}{l} \mathsf{tp}(\mathcal{P}) \to \max \\ \text{i.e. } \mathsf{fn}(\mathcal{P}) \leq \tau \end{array} . \tag{1}$$

Alternatively, we can reduce specificity to guarantee a fixed recall rate:

$$\begin{array}{l} \mathsf{fp}(\mathcal{P}) \to \min \\ \text{i.e. } \mathsf{tp}(\mathcal{P}) \geq \tau \end{array} . \tag{2}$$

In principle, the best solution to optimization tasks is solely determined by the set of used tokens and text corpora we are interested in. In reality, the optimal solution to (1) and (2) might contain so many patterns that we cannot efficiently match them against sentences or learn them from reasonable amount of examples. Hence, one usually considers only a specific sets of patterns that have compact description while solving the tasks (1) and (2). For instance, we can consider only pattern sets that are described by regular expressions. As the latter reduces search space, the resulting solution might be significantly worse than the optimal. Finally, search space of all patterns is usually to large for exhaustive search and we can rely on heuristic algorithms. Hence, the quality of fact extraction methods additionally depends on how we specify pattern sets and what are the complementary search strategies.

The main aim of this work is to find reasonably fine-grained tokens together with a grammar for describing pattern sets such that there exist relatively efficient algorithm for finding approximate solutions to the tasks (1) and (2). We evaluate the results in terms of absolute performance, i.e, how well the resulting extraction algorithm works. For that we use real-world examples of binary relations and text corpora rising from practical applications. Whether the solution is optimal from theoretical perspective is secondary though we try to achieve optimality in that respect, as well.

Main contributions. This work is based on and contains parts of an earlier work named *"Mallipõhine faktituletus tekstikorpustest"* [PL11], written by Sven Laur and Timo Petmanson with equal contributions to theoretical work. The implementation of the prototype application and case studies were supervised by Sven Laur, but carried out by Timo Petmanson.

My main contributions include the theoretical work, that describes the relations as sets of patterns; how to use patterns with basic machine learning methods to train and deploy models for fact extraction; description of the process of active learning, which helps to speed up annotating relations in large corpora. Other contributions include a prototype implementation with plain-text preprocessor, corpus annotator, pattern miner and fact extractor.

Additionally, there are empirical studies about the efficiency of the prototype implementation with several relations and corpora.s

Related work. Most work in field of information extraction has be done and evaluated on English language corpora. For instance, LEILA is designed to pursue binary relations in English text. It uses link grammar to parse the text to and uses semantic structure as features for machine learning methods to represent and generalize the patterns[SIW06, ST95]. Compared to methods like SNOWBALL, which do not use deeper language specific knowledge, the precision and recall of LEILA were respectively around 90% and 38% or more, while the former had precision 34%-48% and recall less than 30% on the benchmark corpora [AG00]. The work was later combined with logic reasoning in systems SOFIE and PROSPERA, which improved the precision and recall further [SSW09, NTW11]. There are systems such as KnowItAll, which extracts facts, concepts and relationships from the web, starting with a seed ontology and a few generic rule templates [ECD$^+$04].

Pattern-based approaches like [CV05, Bri99, BM07] start with seed facts such as “Barack Obama” and “August 4, 1961”. For instance, for birthday relation, patterns like “X was born Y” can be discovered. Using the patterns, new candidate facts are discovered and assessed. These methods are usually able to guarantee good recall, but may not be very precise as the methods may discover patterns like “X died Y”, which will not any more encode a birthday relation. In contrast, our approach learns from specific annotations in texts, thus reducing the possibility to mine ill patterns already in early stages. Similarly, LEILA uses counter-examples as pre-emptive measure to avoid ill patterns [SIW06].

Some methods like [BOS04, Vö05] are designed to work explicitly with instanceOf relations, while our tool can be used to extract any n-ary relation. Systems like YAGO-NAGA [KRSW09] and DBpedia [ABK$^+$07] are geared towards extracting information from Wikipedia. These methods take advantage of the structured information from the pages, contrary to our approach, that does not have this advantage.

Method [CBHM10] uses high-precision patterns to increase the candidate examples and new potential patterns iteratively. Active learning is used in [Sod99, TCM99] and uses frequent human interaction as part of the process of building the model for fact extraction. Similarly, our method is capable of bootstrapping: either extracting high precision or high recall patterns, which can be used to extract new examples or contrary, reduce the selection of sentences, where examples probably do not occur. Combined with human input, this makes annotating larger corpora more efficient as human can evaluate the correctness of the model and also report back misclassified examples.

Related work in Estonian are the named entity recognition tool [Tka10], which can extract person names, locations, organizations and facilities from free-texts. Other notable works contain the language processing tool EST-MORF [Kaa97], constraint-grammar for Estonian [MPM+03] which can be used with VISLCG3 parser. Both tools can be used in combination to extract morphological and syntactical language features.

Roadmap. We define the basics of patterns matching that we will use throughout the work in Chapter 1. We introduce covers and describe, how are patterns related to fact extraction. We talk about the generalization of patterns. Next, we discuss Estonian language specific features and how to use these to abstract single tokens of patterns in Chapter 2. We describe the corpora used in several case studies later in the work. Chapters 3 and 4 introduce pattern mining. We use monotonicity to adapt Apriori algorithm for our tasks. In Chapter 5, we introduce ways to combine patterns with machine learning methods to determine the certainty of matches by patterns. We also carry out case studies using the approach for named entity recognition and extraction birth dates from free-texts. Finally, in Chapter 6, we discuss possibilities to use active learning and human interaction in the process of annotating large corpora.

Chapter 1

Pattern matching

In this chapter, we define patterns and discuss how they are related to each other. We describe a basic set of operations, that allow us to increase or decrease the abstraction level to fit our needs, although we discuss language specific information and related abstraction methods in Chapter 2. Depending on the level of the abstraction and structure of the patterns, we can specify more easily, what types of fragments or sentences the patterns should match. Pattern matching forms the core of this work and is combined with data mining and machine learning in later chapters.

1.1 Patterns and covers

In this work, we consider patterns as sequences of tokens where each token can match a single word. We write such patterns using the sequence notation, e.g. $p = p_1, p_2, \ldots, p_\ell$, where ℓ is the length of the pattern. The length of a pattern p can be also denoted by $|p|$. We refer to a particular token as p_i and to sub-sequence of tokens $p_i, \ldots, p_j$ as $p_{i:j}$ when $i \leq j \leq \ell$. By convention $p_{i:j}$ is empty when $i > j$. We might also mix the notation, such that $p = p_{1:3},\ p_4,\ p_{5:\ell}$ is same as $p = p_1,\ p_2, \ldots,\ p_\ell$.

Patterns can match sentences in texts. For example, a pattern can match a sentence $s = s_1, s_2, s_3, \ldots, s_{10}$ starting from positions 1 and 3. The *cover* of a pattern p contains all positions about all sentences of the text it matches and *support* of a pattern is count of all matches. We define $\mathrm{cover}(p)$ as a set of tuples (s, i, j), where s is the sentence and i is the starting and j the ending position of the match. For example, if pattern p with length of 7 matches sentence $s_1, s_2, s_3, \ldots, s_{10}$ from positions 1 and 3 then $\mathrm{cover}(p) = \{(s, 1, 7),\ (s, 3, 9)\}$.

Let $\mathcal{A}$ and $\mathcal{B}$ be sets consisting of such tuples. Then we can define a *con-*

catenation and *join* operators

$$\mathcal{A}\circ\mathcal{B}=\{(s,i,k):(s,i,j)\in\mathcal{A}\wedge(s,j+1,k)\in\mathcal{B}\}\quad,$$
$$\mathcal{A}\bowtie\mathcal{B}=\{(s,i,j,j+1,k):(s,i,j)\in\mathcal{A}\wedge(s,j+1,k)\in\mathcal{B}\}\quad.$$

Since the concatenation operator takes the matches of two different patterns and imbue them into a single match when matches are consequent, the cover of concatenated patterns $p\circ q$ can be expressed in term of original covers:

$$\mathrm{cover}(q\circ r)=\mathrm{cover}(q)\circ\mathrm{cover}(r)\quad. \tag{1.1}$$

The join operator $\bowtie$ also describes all occurrences of concatenated pattern. However, it additionally describes where one pattern ends and the other continues. The resulting tuple sets can be further joined. The general definition is

$$\mathcal{A}\bowtie\mathcal{B}=\{(s,i_{1:k},\,i_k+1,\,j_{2:\ell})\,:\,(s,\,i_{1:k})\in\mathcal{A}\wedge(s,i_k+1,\,j_{2:\ell})\in\mathcal{B}\}\quad.$$

Also, note that

$$\mathcal{A}\circ\mathcal{B}=\{(s,i_1,i_\ell)\,:\,(s,i_{1:\ell})\in\mathcal{A}\bowtie\mathcal{B}\}\quad.$$

As a single pattern is not enough to describe many relations, we also operate with sets of patterns further referred as *pattern sets*. We define covers for pattern sets as union of all individual covers. For a set of patterns $\mathcal{P}$, the formal definition is following:

$$\mathrm{cover}(\mathcal{P})=\bigcup_{p\in\mathcal{P}}\mathrm{cover}(p)\quad, \tag{1.2}$$

Hence, it is easy to prove that

$$\mathrm{cover}(\mathcal{P}\cup\mathcal{Q})=\mathrm{cover}(\mathcal{P})\cup\mathrm{cover}(\mathcal{Q})$$
$$\mathrm{cover}(\mathcal{P}\cap\mathcal{Q})\subseteq\mathrm{cover}(\mathcal{P})\cap\mathrm{cover}(\mathcal{Q})$$

where the containment can be strict in the second equation. For instance, if two different tokens α and β match the same word then $\mathrm{cover}(\alpha)\cap\mathrm{cover}(\beta)\neq\emptyset$, whereas $\mathrm{cover}(\emptyset)=\emptyset$. Let $\mathcal{P}\circ\mathcal{Q}$ denote the pairwise concatenation of two pattern sets, i.e.,

$$\mathcal{P}\circ\mathcal{Q}=\{p\circ q:p\in\mathcal{P},q\in\mathcal{Q}\}\quad.$$

Then the equation (1.1) assures that

$$\mathrm{cover}(\mathcal{P}\circ\mathcal{Q})=\mathrm{cover}(\mathcal{P})\circ\mathrm{cover}(\mathcal{Q})\quad. \tag{1.3}$$

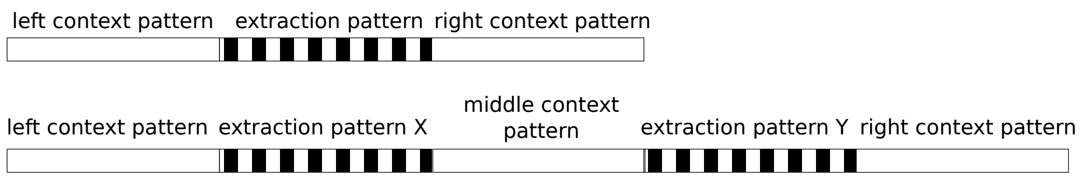


Figure 1.1: Fact extraction with unary (top) and binary (bottom) patterns. Cover elements of extraction patterns are used to denote X and Y examples, where other patterns are used to detect the context.

1.2 Fact extraction

Using the covers of the patterns, we can directly extract the matched sentence fragments as candidate examples for facts. In case of some unary relations, this is exactly what we need to do. For instance, if we want to detect dates, we might have a pattern

numeric *jaanuar* numeric

that can easily detect sentence fragments such as "21. jaanuar 1956" etc. But in case we want to restrict our patterns to birth dates, we need to consider the context of the date. The sentence

Jüriöö ülestõus algas 23. aprillil 1343.
St. George's Night Uprising began on April 23 in 1343.

does not express the relation, while sentence

Toomas Hendrik Ilves sündis 26. detsembril 1953 aastal Stockholmis.
Toomas Hendrik Ilves was born on December 26 in 1953 in Stockholm.

does. If pattern p extracts the dates and pattern α is relevant prefix and β suffix, we must extract only the fragments, that belong to $\mathrm{cover}(\alpha) \bowtie \mathrm{cover}(p) \bowtie \mathrm{cover}(\beta)$ to detect only the birth dates. Figure 1.1 depicts how context and extraction patterns can be used to detect relevant matches.

We do not use separate patterns for context areas, but just specify which tokens correspond to entities we intend to extract. Let $p_1, \ldots, p_\ell$ be the pattern and let $p_{j:k}$ be the *extraction area* corresponding to the entity X. Then the corresponding *extraction mask*

$$\mathrm{cover}_X(p) = \{(s, (i_{2j-1}, i_{2k})) : (s, i_{1:2\ell}) \in \mathrm{cover}(p_1) \bowtie \cdots \bowtie \mathrm{cover}(p_\ell)\}$$

lists all sentences together with start and end positions for X. For binary and ternary patterns, extraction masks are defined analogously. For instance, if p_3

and p_6 are the extraction tokens corresponding to X and Y and the pattern matches a sentence s from positions 1 and 3, then

$$\mathrm{cover}_{XY}(p) = \{(s,(3,3),(6,6)),(s,(5,5),(8,8))\} \ .$$

We use parenthesis to group the extraction areas for X and Y as they can match more than one token. Analogously, we can define extraction mas for pattern sets, where for binary relations the formula is

$$\mathrm{cover}_{XY}(\mathcal{P}) = \bigcup_{p\in\mathcal{P}} \mathrm{cover}_{XY}(p) \ .$$

1.3 Comparing patterns

Patterns can be compared on the syntactic and semantic level, i.e., how they are written down and what relations do they extract. On the syntactic level, we define $p \preceq q$ as a transitive closure of the following two rules. First, any consequent subsequence q is less specific than the entire pattern p:

$$p = \alpha \circ q \circ \beta \qquad \Rightarrow \qquad p \preceq q \ . \tag{1.4}$$

The extraction areas of p and q must be of same width and by removing α and β, this must not change. We require this to ensure that more generic patterns can always extract everything the more specific pattern does. Second, a pattern becomes less specific if we substitute its sub-pattern with more generic alternative of same length:

$$\alpha \preceq \beta \wedge |\alpha| = |\beta| \qquad \Rightarrow \qquad \alpha \circ p \preceq \beta \circ p \ . \tag{1.5}$$

To apply this rule, we must specify how to compare individual tokes. For example, a generalized token can allow any word type, where specific token might require exact type. The latter is language and application specific and is discussed in Chapter 2.

On the semantic level, we use a shorthand $p \sqsubseteq q$ to denote that all relations between entities revealed by the pattern p are also revealed by the pattern q. If patterns p and q are meant to extract a single entity X, the latter means:

$$p \sqsubseteq q \qquad \Leftrightarrow \qquad \mathrm{cover}_X(p) \subseteq \mathrm{cover}_X(q) \ . \tag{1.6}$$

The relation is defined analogously for binary and ternary patterns. It is easy to see that syntactic relations between patterns are carried over to extraction masks:

$$p \preceq q \qquad \Rightarrow \quad p \sqsubseteq q \ . \tag{1.7}$$

However, whether the converse holds depends on the text corpus.

Proposition 1.3.1. *Let p and q be valid patterns. Then $p \preceq q$ implies that $p \sqsubseteq q$ provided that $p \preceq q$ implies $\mathrm{cover}(p) \subseteq \mathrm{cover}(q)$ for individual tokens.*

Proof. For the formal proof, we must do structural induction over the way the relation $p \preceq q$ is defined. Let us now observe the first rule. If $p = \alpha \circ q \circ \beta$ then each match of p contains also a match of q. Since the prefix α and the suffix β contain no extraction tokens the corresponding extraction masks for p and q are the same. For the second rule, note that if $\alpha \preceq \beta$ and $|\alpha| = |\beta|$ then $\alpha \preceq \beta$ only due to the application of the second rule. Thus, we can do the induction over the number of times the second rule is applied. If the rule is applied the first time then α and β are tokens. By the assumption $\mathrm{cover}(\alpha) \subseteq \mathrm{cover}(\beta)$ and consequently $\mathrm{cover}(\alpha \circ p) \subseteq \mathrm{cover}(\beta \circ p)$. For the longer patterns α and β, we can similarly show $\mathrm{cover}(\alpha \circ p) \subseteq \mathrm{cover}(\beta \circ p)$. To complete the proof, note that for the same match, locations of extracted tokens must coincide, since extraction areas are in the same locations in both patterns and each token matches only a single word. □

The main reason why the converse implication does not generally hold are implicit restrictions on the sentences. For instance, syntactically incomparable patterns $\alpha \circ p$ and $p \circ \beta$ might have the same cover in text corpus if the pattern p occurs as $\alpha \circ p \circ \beta$ in all sentences. As a result, syntactic information alone is not enough to decide when extraction masks are subsets of each other.

Similarly, we can define ordering between patterns sets in terms of extraction masks. For unary pattern sets, the corresponding definition is following:

$$\mathcal{P} \sqsubseteq \mathcal{Q} \qquad \Leftrightarrow \qquad \mathrm{cover}_X(\mathcal{P}) \subseteq \mathrm{cover}_X(\mathcal{Q}) \ . \tag{1.8}$$

On the syntactic level, we say that pattern set $\mathcal{Q}$ is more generic than $\mathcal{P}$, if for each pattern $p \in \mathcal{P}$ there exists a more generic pattern $q \in \mathcal{Q}$. In formal terms,

$$\mathcal{P} \preceq \mathcal{Q} \qquad \Leftrightarrow \qquad \forall p \in \mathcal{P}\ \exists q \in \mathcal{Q} : p \preceq q \ . \tag{1.9}$$

From the relations (1.8), (1.9) and (1.7) we can directly conclude that syntactic relations between pattern sets are carried over to extraction masks:

$$\mathcal{P} \preceq \mathcal{Q} \qquad \Rightarrow \qquad \mathcal{P} \sqsubseteq \mathcal{Q} \ .$$

As before, the converse implication does not hold in general.

1.4 Overlapping matches of extraction areas

Note that Definition (1.6) requires exact matches in terms of extracted entities. For instance, if a pattern p extracts

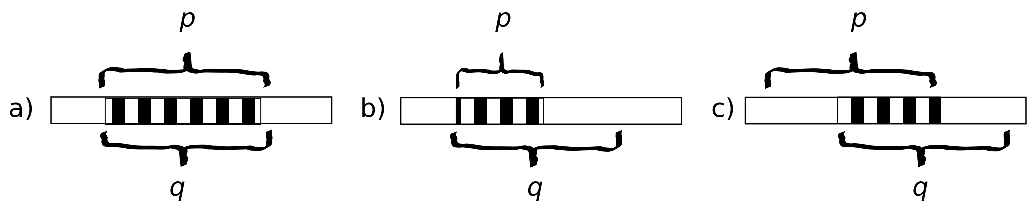


Figure 1.2: Overlapping matches of extraction areas of pattern p and q. Matches are exact (left), one is submatch of another (b), matches have common overlap (right).

President Barack *<u>Obama</u>* *meets president Ilves in Kadriorig.*

and a pattern q extracts

President *<u>Barack Obama</u>* *meets president Ilves in Kadriorig.*

from the same sentence, then one match is a submatch of another. To be more exact, the overlapping can occur in three different ways: a) the extracted areas match; b) one match is subarea of another match; c) two matches have common overlaps, but neither is a submatch of another. See Figure 1.2.

We explicitly require exact matches as our formulation of pattern mining discussed in Chapter 3 will not work with partially overlapping extraction areas. However, we still deal with these by using pattern sets. If we need to represent both the matches of 'Barack Obama' and 'Obama', we create separate patterns with extraction areas of required width and make syntactic and semantic comparisons at pattern set level using Equations 1.8 and 1.9.

The question, which of the two sentence fragments is more correct and should be actually used, is an independent question and involves classifier methods discussed in Chapter 5. For a patterns $\mathcal{P}$ with a classifier $\mathcal{C}$ and a corpus, we get $n = |\mathcal{P}|$ covers by matching the patterns in the corpus. The union of the covers

$$\mathrm{cover}_X(\mathcal{P}) = \bigcup_{p \in \mathcal{P}} \mathrm{cover}_X(p)$$

can contain overlapping cover elements. Analogously, this can happen for binary and ternary patterns. However, in natural language, it is typically not possible to encode several examples of a single relation, that are overlapping in a sentence. Therefore, we need a way to deal with the overlaps and decide, which examples should we keep.

Let $\Pr[e]$ be the probability that cover element $e \in \mathcal{C}$ really encodes a particular relation. The probability can be proportional to the number of patterns, that yield the element or it can be a probability assigned by a machine learning model, that we discuss in Chapter 5. For each set of overlapping cover

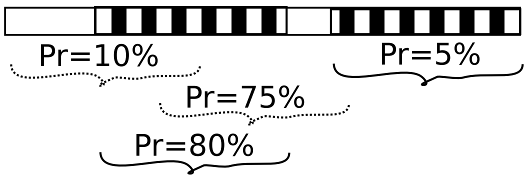


Figure 1.3: Greedy overlap resolution. From a set of of overlapping extraction mask elements, we accept the one with highest certainty, remove the conflicting elements continue similarly. Dashed lines represent the removed elements and striped boxes represent mask elements accepted in the process.

elements, we can just greedily accept the ones starting with highest probabilities and each step remove the overlapping elements until no overlaps occur. See Figure 1.3 for an example.

1.5 Statistical measures for evaluation

So far we have mentioned several statistical measures like fp-rate, fp-rate, sensitivity etc. These statistical measures can be described as equations of the number of true positives, false positives, true negatives and false negatives produced by the patterns. The positive examples are the extraction mask elements, that express the relation. Consider a sentence

President <u>*Barack Obama*</u> *meets president Ilves in Kadriorig.*

If the pattern extracts "Barack Obama," then we can count it as as true positive match. If the pattern extracts "President Barack Obama", we also should count it as a true positive match as the correct example is a subphrase of the extracted text fragment. If the pattern only extracts "Obama", we count it as a false positive, because it does not cover the whole example and information is lost. Note that this is completely valid choice as it does not break how the syntactic and semantic relationships between patterns are defined. In case of n-ary patterns, we require this for all extraction tokens.

The false negative examples are those, which encode a relation, but are not detected by the pattern. We can detect false negatives only in testing phase with annotated data. True negatives are examples, which do not encode the relation and are not detected. The problem is that depending on the arity of the patterns, the widths of the extraction areas and length of the sentence, there can be many true negative examples. In our case, we ignore that fact and assume that the number of true equals to the number of sentences, where no single match or example occurs. It is not reasonable to try to identify all possible true negative match combinations.

Let *TP*, *FP*, *TN* and *FN* represent the respective counts of true positives, false positives, true negatives and false negatives. One important measure is recall, which is the proportion of true positive examples a pattern was able to detect in the corpus:

$$\mathsf{tp}(p) = \mathsf{sensitivity}(p) = \mathsf{recall}(p) = \frac{TP}{TP + FN} \quad . \tag{1.10}$$

High sensitivity (recall) is very important, if we want to discover all true positive examples. However, with high sensitivity, we might have many false positives. The amount of false positives among matches is determined by precision, which is the number of true positives among all the matches of a pattern:

$$\mathsf{precision}(p) = \frac{TP}{TP + FP} \quad . \tag{1.11}$$

Closely related to precision is false discovery rate (FDR)

$$\mathsf{fdr}(p) = 1 - precision = \frac{FP}{TP + FP} \quad .$$

False positive rate is the proportion of incorrect matches among all possible invalid matches:

$$\mathsf{fp}(p) = 1 - \mathsf{specificity}(p) = \frac{FP}{FP + TN} \quad . \tag{1.12}$$

Calculating this metric requires that we know the amount of true negatives, which we agreed, was the number of true negative sentences. However, if a pattern yields several false positive matches in a single sentence, we count these as two distinct matches, thus increasing the false positive rate. This way, we always over-estimate the metric, thus real fp-rate rate is possibly lower.

False negative rate tells the proportion of examples, that were not discovered:

$$\mathsf{fn}(p) = 1 - \mathsf{sensitivity}(p) = \frac{FN}{TP + FN} \quad . \tag{1.13}$$

This metric can only be evaluated, if we have fully annotated corpus or we could use human interaction.

1.6 Estimations on unannotated data

If we are applying the patterns and model on unannotated data, we can use human interaction as a part of process to evaluate the metrics. To estimate precision, we can draw a sample with replacement of n matches and let the user determine the amount of true positives. The number of true positives follows a Binomial distribution, where p is the precision and estimate $\hat{p}$ is the proportion of true positives in user sample. Given that n is large enough (about more than 30 samples) and the probability is not too extreme (near to zero or one), then Binomial distribution approximates to Normal distribution, such that $\hat{p} \sim N(n\hat{p},\, n\hat{p}(1-\hat{p}))$ and we can provide standard confidence interval

$$\hat{p} \pm \lambda_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\ ,$$

where $1-\alpha$ is the confidence level and $\lambda_{1-\alpha/2}$ is the $1-\alpha/2$ percentile of standard normal distribution. However, the performance of standard interval can be chaotic and poor as shown in [BCD01] and it is better to use another interval such as Wilson interval

$$\frac{\hat{p} + \frac{1}{2n}\lambda^2_{1-\alpha/2} \pm \lambda_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{\lambda^2_{1-\alpha/2}}{4n^2}}}{1 + \frac{1}{n}\lambda^2_{1-\alpha/2}}\ .$$

To estimate false positive rate, we need to draw a sample with replacement from all positive and negative matches and determine the total number of false positives and true negatives n, where m is the amount of false positives. The estimate is $\hat{fp} = \frac{m}{n}$ and confidence intervals can be calculated analogously as described above. Similarly, we can estimate other metrics.

Chapter 2

Tokens and language specific information

In Chapter 1, we introduced syntactic and semantic relationships between patterns. In this chapter, we describe how we can use language specific information as a mechanism of abstracting individual tokens i.e words and named entities of the Estonian language.

2.1 Word attributes and values

To be able to compare the patterns at syntactic level, we must specify, how to compare the individual tokens. Tokens represent single words or named entities, which can be simply written down as character strings. However, this is not the only possible representation. Using language processing tools, we can extract the canonical form of the word and augment it with language specific information such as the case, plurality and mode of the word. The information is stored as a set of morphological and grammatical attribute values. We use dot notation $p.attr$ to mark the value of attribute $attr$ of a token p.

The exact number of attributes can vary depending on our needs and abilities to perform sentence-level analysis. Current availability of NLP tools allows us to lemmatize and extract morphological and syntactic attributes of words with feasible accuracy [Kaa97]. Additional syntactic analysis can be performed using a constraint grammar parser VISLCG3 using Estonian grammar [MPM+03].

We decided to use 12 attributes extracted from the analysis: lemma, word type, verb type, case, plurality, mode, genus, presens, time, comparisons, capitalization, syntactic role in the sentence. Attribute information is structured as a set of trees and is derived from common attribute value categorization in Es-

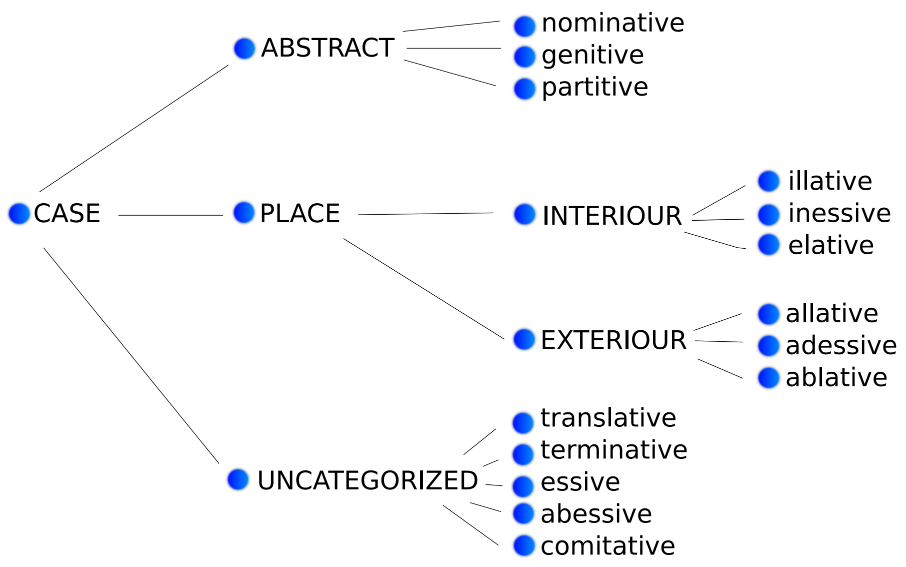


Figure 2.1: The word case hierarchy. Specific cases are the leaves of the tree and are written in lowercase. Parent nodes represent the generalizations of the cases.

tonian language theory [EKM+93, EKM+95]. For example, there are 14 cases in Estonian: nominative, genitive, partitive are *abstract* cases. Illative, inessive and elative are so called *interior local* cases. Allative, adessive and ablative are *exterior local* cases. And the rest are *uncategorized*. See Figure 2.1.

For defining comparison relation between tokens, we need to define how to compare attribute values. For any attribute, let $a \prec b$ denote that an attribute value b is an ancestor of a in the respective attribute value tree. From Figure 2.1, we see that `partitive` $\prec$ `abstract` and `genitive` $\not\prec$ `place`.

Different types of words have different attributes. In such cases we denote the missing attribute values as $\top$ instead, which represents the root values of these particular attributes. By this we postulate that any attribute value is suitable, since $a \preceq \top$ for any valid attribute value a. The generalization to the token level is natural. A relation $p \preceq q$ holds if the corresponding values $p.attr \preceq q.attr$ for all attributes $attr$. We acknowledge that there could be other more fine-grained ways to define ordering between tokens. However, the latter requires more deeper understanding of Estonian language.

A token q *matches* a token p if and only if $p \preceq q$. For example, a token t with $t.case =$ `abstract`, $t.plurality = \top$ matches any word in any abstract case (nominative, genitive, partitive) and allows both singular and plural forms. Note that the assumption of Proposition 1.3.1:

$$\mathrm{cover}(p) \subseteq \mathrm{cover}(q) \quad \text{for all tokens} \quad p \preceq q$$

is trivially satisfied as $\preceq$ is transitive relation.

The attribute value trees are not very deep for some of the attributes. For example, plurality tree has depth of 2 and has only three nodes: root $\top$ and its direct children *singular* and *plural*. The canonical form has as many nodes as there are words in a language plus the root node $\top$, which is a direct parent of all other nodes.

The tokens can be defined for other languages beside Estonian as well. The only requirement is the availability of suitable NLP tools that can extract the attributes and there exists meaningful ways to generalize them.

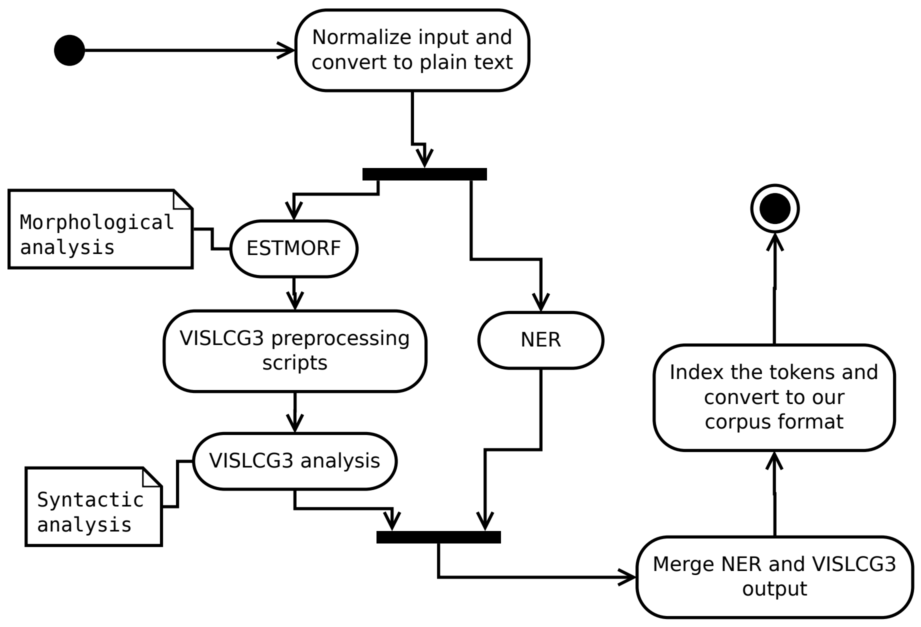
2.2 Text corpora

In practice, we cannot apply our patterns on plain texts without processing them first. The reason is that we need to extract language specific features, which can be used as the attributes of the tokens. As a result, the sentences become lists of tokens, where each token is a set of attribute/value pairs, where a particular value is a leaf node of a respective attribute value tree.

We decided to store the processed text corpora as entity-relationship databases. Although it is not as convenient for string processing as simple lists, it makes it simpler to build full indexes for fast cover retrieval of patterns and individual tokens. Additionally, we can include the definitions of attribute value trees, so that calculating covers of tokens with abstracted attribute values can be formulated as SQL queries. It is also convenient to store additional information such as annotations for different relations. Finally, using an external database handles memory management, storage and caching of large corpora and indexes.

2.2.1 Data preprocessing pipeline

As discussed in Section 2, we use tools ETMORF and VISLCG3 to do morphological and syntactical analysis. Optionally, we can do named entity recognition on the input text and use its annotations as an extra attribute for the tokens. Another possibility is to use the information to concatenate tokens representing names like *Rauno Thomas Moss* or locations like *Rio de Janeiro* into single tokens instead. The full processing stack is given in Figure 2.2.1.
First step is data conversion to plain text in case of HTML content and normalization: unify quotation marks, force UTF-8 encoding, put each sentence on a separate line. Next, we do morphological and syntactical analysis in parallel with named entity recognition due the dependencies of these tools. Last, we merge their output and convert it to a entity-relationship database usable



Normalize input and convert to plain text
Morphological analysis
ESTMORF
VISLCG3 preprocessing scripts
NER
VISLCG3 analysis
Syntactic analysis
Index the tokens and convert to our corpus format
Merge NER and VISLCG3 output

Figure 2.2: Data preprocessing pipeline. NER step is optional.

by our data mining scripts. See Appendix A for the details of the prototype implementation for more information.

2.3 Corpora used in this work

In this section, we introduce three different text corpora, which are later used in different case studies for evaluation of our methods. In general, the methods are applicable to any kind of textual data, but their performance depends on data and language to be applied. Simple sentences that express facts and meanings in a straightforward manner are best to work with, whereas long sentences with complex grammar are harder to analyze. Hence, we use three different free-texts: short and informative Twitter feeds, Wikipedia articles, ordinary news articles from Postimees. These corpora contain sentences of slightly different linguistic complexity and are thus suitable for practical benchmarks of our methods.

2.3.1 Twitter corpus

One of our initial goals was to build a tool for extracting binary relation that connects events to their locations. As such, it would allow us to scan the news-feed and cluster articles based on event types and physical locations or seek texts that are related to a particular place. We will refer to this relation as event location relation onwards. In principle, we could use all three data sources for training. However, the occurrence rate of event location is very low in Wikipedia entries and newspaper articles and in many cases the relation is spread over several sentences. The occurrence rate is much higher in Twitter tweets, since they are short and independent units of communication with no room for irrelevant information.

To make our life easier, we chose sources that broadcast a lot of information about events and are likely to use newspaper language. More precisely, we gathered news headings from following Twitter accounts:

minut_ee	postimees	TlnPostimees	TrtPostimees
DelfiEE	Oleht	ParnuPostimees	estonia_ee
teaduspark	TallinnaYlikool	viljandi	Kutser
Tarbija24	errsport	err_ee	StenbockiMaja
JarvaTeataja	MinuEesti	reporteronline	Finantsportaal
LoovEesti	kumuartmuseum	Soomaa_Parnu	valiskomisjon
Linnateater	eestekspress	valismin	Riigikogu

The date of the download was 4th of November 2011 and it contained about 2000 most recent headlines. In general, the sentences of the corpus were quite straightforward, although not necessarily very simple in structure.

The Twitter corpus contained 173 examples (in about 8% of all sentences) of event location relation. All the positive examples (event and location) were annotated manually. In many cases, the extraction tokens were not single words, but named entities such as "Viljandi Kultuurikeskus". However, our linguistic preprocessing tools are not perfect and yielded some errors in the resulting corpus. For example, some locations were not recognized as entities. In such cases, we explicitly annotated the less ambiguous token as the extraction token. For example, if phrase 'Tallinnas Kuku Klubis' was not recognized as location, we chose 'Tallinn' as the extraction token. Similarly, for events like 'Viljandi meistrivõistlused võrkpallis', we chose 'meistrivõistlused' as the extraction token.

Additionally, we made a 50/50 version of the corpus, where we kept all the sentences having the event location relation plus chose a random sample of the same amount of sentences not having the relation. This version had 340 sentences where 50% of true positive sentences. As such the corpus is strongly enriched in event location relation. We used it to test pattern mining and machine learning algorithms, as it is easier to train machine learning algorithms on balanced data. We used the original (not enriched) corpus to test active learning strategies.

2.3.2 Wikipedia corpus

As the second goal, we tested a binary birthday relation, which connects person names to their birth dates. This relation is a common test target in several information retrieval applications for English [SIW06]. Also, it is rather common to use Wikipedia articles as a test benchmark. The birthday relation is less ambiguous compared to event location and thus much simpler to detect. Also, we can compare our results with the results for the English language.

We compiled the corpus by downloading 100 bibliographies from the Estonian Wikipedia about famous people. Again, the annotation of extraction tokens (name and birthday) was done manually. We used year of the birth date and surname of the person as extraction tokens, in case full timestamps or names were not recognized by the NER toolchain. The language of the articles was in general simpler than the language of Twitter corpus.

About 14% of the sentences of the corpus contained the birthday relation, where we had total of 116 annotations in 813 sentences. Again, we made a 50/50 version of the corpus, where we kept all the true positive sentences with a random sample of sentences not having the relation with size equal to true positive sentences. The 50/50 version of the corpus had total of 228 sentences.

As with twitter corpus, we used the 50/50 version of the corpus in machine learning experiments and the original version in active learning experiments. Differently from event location it is rather easy to achieve enrichment ratios close to 20% by just picking sentences containing dates from the bibliographic Wikipedia entries. Hence, we did not build the corpus where the ratio of positive sentences is close actual occurrence rate.

2.3.3 Postimees Online corpus

We also studied how well our methods can be applied for Named Entity Recognition (NER). The goal in NER is to extract all sequences of tokens that are either persons, organizations or location. As such, it is can be viewed as extraction of various unary relations: person, organization and location. This problem is well studied for Estonian language and there exists a good extraction tool [Tka10] together with manually annotated data used to train and evaluate this tool. We use thus data as a benchmark to test the performance of our methods for unary relations.

Note that our aim is not to come up with an alternative for NER, as the current solution is already quite elaborate. Instead, we study mainly two aspects. First, how well do extraction patterns work and what is the coverage of individual patterns. Second, does the performance of the NER tool increases significantly if we include patterns as features. In particular, can we find patterns that resolve ambiguities, e.g. discriminate between organizations and locations.

The benchmark corpus consists of news articles published by Postimees Online, which is a very popular Estonian news portal. Sentences from these articles have more difficult structure than the ones in Twitter and Wikipedia corpus. As before, all sentences are manually annotated. We used these annotations as golden truth to create a balanced corpus with 50/50 split for each entity type. However, we did not include the entire corpus. Instead, we used only a random sample of 20% of these sentences to make it small enough for our prototype implementation.

Also, not all sentences were considered for sampling. We could not use the morphological information of the original corpus as its format was too different from our requirements. Instead, we reconstructed original sentences and reanalyzed them with our own data preprocessing stack. As a result, about 5% of the sentences had to be excluded as our pipeline treated some punctuation differently from the original corpus and made annotations of these particular sentences invalid.

Chapter 3

Pattern mining tasks

In this chapter, we are going to discuss, how to extract meaningful set of patterns from corpora, which would cover specific relations, we are interested in. One possibility to do that would be to define the patterns manually, but this requires good understanding of the language and generally it is impossible to take into account the language of different corpora. While news corpora might use one type of language, medical texts contain different type of language. The usage of words, expressions and vocabulary can differ from corpus to corpus. Thus, an alternative approach is required.

The reason we need to do pattern mining is that we cannot cover all possible examples simply by manual annotation. Therefore, we annotate a number of examples and use these examples to extract patterns with desired abstraction level. Higher abstraction allows to detect new unseen examples with certain probability, where details may differ from examples we have already seen.

We are going to discuss semi-automated means to extract the patterns from a fully or partially annotated training corpus. The annotation process is manual, but it will define the extraction tokens of examples, that the patterns should be able to extract. This information can be processed with the computer algorithms. We can use the annotations to define initial patterns and use data mining methods to generalize the information, in hope that the resulting model will cover most examples with acceptable precision.

The formal definitions of our pattern mining tasks are to find a meaningful set of patterns, which have either good sensitivity or good specificity and help us to find approximate solutions to global optimization tasks (1) and (2). Each pattern represents a subset from all possible cover elements and we want to find a minimal number of patterns covering the true positive cover elements, while still satisfying the false positive rate or false negative rate criteria. These two optimization tasks are closely related to the set cover and knapsack pack-

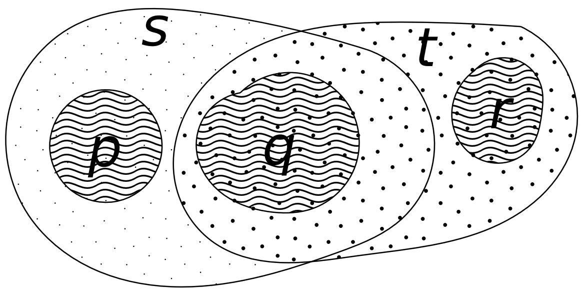


Figure 3.1: Venn diagrams for covers of patterns p, q, r, s, t, where $p, q \preceq s$ and $q, r \preceq t$, if $s \not\preceq t$ and $s \not\succeq t$.

ing problems and are thus generally infeasible to solve. Hence, we first consider all patterns with a low rate of false positives (further referenced as *fp-safe patterns*)

$$\mathcal{P}_{\text{fp}}(\tau) = \{p : \mathsf{fp}(p) \leq \tau\} \tag{3.1}$$

and patterns with high rate of true positives (*fn-safe patterns*)

$$\mathcal{P}_{\text{fn}}(\tau) = \{p : \mathsf{tp}(p) \geq \tau\} \quad . \tag{3.2}$$

These patternsets have a property that $\mathsf{fp}(\mathcal{P}_{\text{fp}}(\tau)) \leq n\tau$ and $\mathsf{tp}(\mathcal{P}_{\text{fn}}(\tau)) \geq \tau$, where n is the number of patterns. We can give guarantees about false positive or false negative rates for the whole pattern set. In case of $\mathcal{P}_{\text{fp}}$ patterns, we get matches that are likely true positives and we can get new examples. Using $\mathcal{P}_{\text{fn}}$ patterns, we can guarantee with certain probability, that the non-matched sentences do not encode the relation, thus making it possible discard many negative sentences. This is important, when we have annotated a small subset of the corpus and want to avoid looking through many possibly negative examples in the rest of the sentences.

3.1 Search strategies based on monotonicity

Due to Proposition 1.3.1, the rates of false and true positives are monotonously increasing:

$$p \preceq q \qquad \Rightarrow \qquad \mathsf{fp}(p) \leq \mathsf{fp}(q) \enspace , \tag{3.3}$$
$$p \preceq q \qquad \Rightarrow \qquad \mathsf{tp}(p) \leq \mathsf{tp}(q) \enspace . \tag{3.4}$$

To be precise, the rate of false positives and true positives is defined only if we fix a text corpus, where all occurrences of a relation to be sought are marked. The latter is often infeasible or too costly in practice and we have to be satisfied with incomplete labeling where some occurrences are unmarked. As a result, we can only approximate true values of $\mathsf{fn}(p)$ and $\mathsf{tp}(p)$ by using available information. However, the implications hold for approximations, as well.

Note that in the process of estimating false positive or true positive rates for the patterns, we need to match the patterns in a corpus and determine, which extraction mask elements correspond true positive and false positive matches. Additionally, we need to consider non-matched examples and determine the true negatives and false negatives. How it is done in detail, is discussed in Section 1.5.

As the implications (3.3) and (3.4) are analogous to the condition used in frequent pattern mining [Goe03, HCXY07], we can take over many methods form this subfield. In particular, limit the search space. To find fp-safe patterns $\mathcal{P}_{\mathrm{fp}}(\tau)$, we can start from complex patterns and generalize them until the rate of false positives is still below τ. To find fn-safe patterns $\mathcal{P}_{\mathrm{fn}}(\tau)$, we start form generic patterns and specialize them until the rate of true positives is still above τ.

Also, note that not all patterns are equally informative in the sense that generic patterns cover all the matches of more specific patterns. In particular, it is enough to store only *maximal patterns*

$$\mathcal{M}_{\mathrm{fp}}(\tau) = \{p \in \mathcal{P}_{\mathrm{fp}}(\tau) : \forall q \succ p : \mathsf{fp}(q) > \tau\} \quad , \tag{3.5}$$

$$\mathcal{M}_{\mathrm{fn}}(\tau) = \{p \in \mathcal{P}_{\mathrm{fn}}(\tau) : \forall q \prec p : \mathsf{tp}(q) < \tau\} \quad , \tag{3.6}$$

as the remaining patterns in $\mathcal{P}_{\mathrm{fp}}(\tau)$ are just all specializations of maximal patterns $\mathcal{M}_{\mathrm{fp}}(\tau)$ and the remaining patterns in $\mathcal{P}_{\mathrm{fn}}(\tau)$ are just all generalizations of maximal patterns $\mathcal{M}_{\mathrm{fn}}(\tau)$. Figure 3.1 depicts covers of five patterns p, q, r, s and t, where s and t cover everything patterns p, q and r do. Thus in process of mining fp-safe patterns, $\mathsf{fp}(s) \geq \mathsf{fp}(p), \mathsf{fp}(q)$ and $\mathsf{fp}(t) \geq \mathsf{fp}(q), \mathsf{fp}(r)$. Similarly, in process of mining fn-safe patterns $\mathsf{tp}(s) \geq \mathsf{tp}(p), \mathsf{tp}(q)$ and $\mathsf{tp}(t) \geq \mathsf{tp}(q), \mathsf{tp}(r)$.

A more fine-grained characterization of $\mathcal{P}_{\mathrm{fp}}(\tau)$ and $\mathcal{P}_{\mathrm{fn}}(\tau)$ is given by considering closed patterns. A pattern p is *closed under* $\mathsf{fp}(\cdot)$ *measure* if $\mathsf{fp}(p) < \mathsf{fp}(q)$ for all generalizations $q \succ q$. A pattern p is *closed under* $\mathsf{tp}(\cdot)$ *measure* if $\mathsf{tp}(p) > \mathsf{tp}(q)$ for all specializations $q \succ q$. Non-closed patterns are redundant in the sense that there is either more general or more special pattern that has the same measure. In particular, it is easy to verify that solutions to the original optimization tasks (1) and (2) consist of closed patterns. Hence, we can

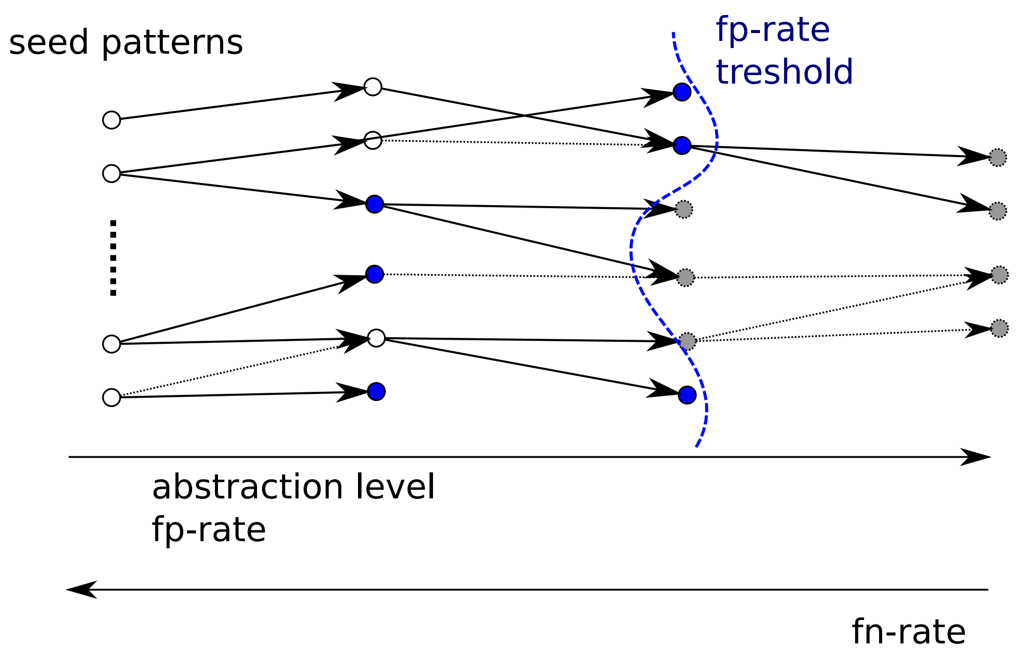


Figure 3.2: The generalization steps when mining *fp-safe* patterns with Algorithm 1. Blue dots represent maximal patterns, gray dots are infrequent (too generic) patterns, white dots are frequent, but not maximal patterns. Arrows depict the generalization DAG of the seed patterns, where solid ones show the actual search tree and dashed ones the edges that will not be traversed.

consider only closed patterns:

$$\mathcal{C}_{\mathsf{fp}}(\tau) = \mathcal{P}_{\mathsf{fp}}(\tau) \setminus \{p : \exists q \succ p : \mathsf{fp}(q) = \mathsf{fp}(p)\} \quad , \tag{3.7}$$

$$\mathcal{C}_{\mathsf{fn}}(\tau) = \mathcal{P}_{\mathsf{fn}}(\tau) \setminus \{p : \exists q \prec p : \mathsf{tp}(q) = \mathsf{tp}(p)\} \quad . \tag{3.8}$$

Of course, the claim is true only if we indeed have completely labeled corpus. Otherwise, we can only approximate $\mathsf{fp}(p)$ and $\mathsf{tp}(p)$ and thus closed patterns under incomplete labeling do not correspond to closed patterns under complete labeling.

3.2 Algorithm for finding fp-safe patterns

To traverse all fp-safe patterns we have to start from a list of seed patterns and then traverse the lattice of all patterns by generalizing patterns until the rate of false positives goes over the threshold τ. To evaluate fp measure, we have to fix a corpus consisting of negative examples—set of sentences that do not encode the relation we seek. As a first approximation, we can take a random sample $\mathcal{N}$ from the text corpus $\mathcal{C}$, we are interested in. In most cases, sentences that indeed encode relations are rare and we overestimate the true rate of false positives. Indeed, let $\mathsf{fn}_{\mathcal{N}}(p)$ and $\mathsf{fn}^{\circ}_{\mathcal{N}}(p)$ be the reported and true rate of false negatives over the sample $\mathcal{N}$. Then

$$\mathsf{fn}_{\mathcal{N}}(p) = \mathsf{fn}^{\circ}_{\mathcal{N}}(p) + \mathsf{tp}^{\circ}_{\mathcal{N}}(p) - \mathsf{tp}_{\mathcal{N}}(p) \quad . \tag{3.9}$$

where $\mathsf{tp}_{\mathcal{N}}(p)$ and $\mathsf{tp}^{\circ}_{\mathcal{N}}(p)$ are the reported and true rate of true positives. By the law of large numbers sample means converge to true means over the entire corpus $\mathcal{C}$ and consequently

$$\mathsf{fn}_{\mathcal{N}}(p) \approx \mathsf{fn}_{\mathcal{C}}(p) + \mathsf{tp}_{\mathcal{C}}(p) - \mathsf{tp}_{\mathcal{N}}(p) \tag{3.10}$$

where $\mathsf{tp}_{\mathcal{C}}(p)$ is bounded by the actual occurrence rate σ of the relation we are interested in. In other words, if $\sigma \leq \tau$ there is no need for correction. Otherwise, we should either estimate offset term $\mathsf{fp}_{\mathcal{C}}(p)$ by estimating σ or label enough positive examples so that $\sigma - \mathsf{tp}_{\mathcal{N}}(p) \leq \tau$.

Consequently, we still have to find all patterns satisfying $\mathsf{fp}_{\mathcal{N}}(p) \leq \tau$ where τ might be different from the initial threshold if we apply correction. Algorithm 1 depicts the resulting depth-first search algorithm for mining maximal fp-safe patterns. Depth-first search is preferable to breath-first search in terms of memory usage, as a maximal fp-safe pattern might have huge number of specializations, which we can side-step during the depth-first search. Second, we can use heuristic generalization methods ∇_i for finding candidate patterns several levels above current pattern. These methods can utilize any information that is available when they are invoked, like the list of known maximal patterns $\mathcal{M}$ so far or rates of false positives. In particular, let the set of direct descendants be denoted as

$$\nabla_{\circ}(p) = \{q : p \prec q \wedge \nexists r : p \prec r \prec q\} \quad .$$

Then it is straightforward to prove the following claim.

Proposition 3.2.1. *If at any step the candidate list $\mathcal{Q}$ contains all direct descendants $\nabla_{\circ}(p)$ then Algorithm 1 recovers all patterns from $\mathcal{M}_{fp}(\tau)$ that are generalizations of seed patterns $\mathcal{Q}_0$.*

Proof. Clearly, any descendant of a pattern $p \in \mathcal{Q}_0$ can be reached by going through a list of direct descendants $p = p_0 \prec p_1 \ldots \prec p_\ell = q$. Now if $\mathsf{fp}_{\mathcal{N}}(q) \leq \tau$ then monotonicity of false positive rate (3.3) assures that $\mathsf{fp}_{\mathcal{N}}(p_i) \leq \tau$ for $i \in \{0, \ldots, \ell\}$. By the assumption p_{i+1} is always in the descendant set Q for p_i and thus algorithm reaches q. If q is a maximal pattern then $\mathsf{fp}_{\mathcal{N}}(r) > \tau$ for any of its descendants and thus q is included into the set $\mathcal{M}$ as $\mathcal{M}_1 = \emptyset$ for q. $\square$

```
Input: Initial set of patterns $\mathcal{Q}_0$.
Output: The set of all maximal patterns $\mathcal{M}_{\mathsf{fp}}(\mathcal{Q}_0, \tau)$ that are descendants
        of $\mathcal{Q}_0$.
Auxiliary functions: List of pattern generalization operators $\nabla_1, \ldots, \nabla_k$.
MaxFPSafePatterns($\mathcal{Q}_0, \tau$)
begin
    $\mathcal{M} = \emptyset$
    for $p \in \mathcal{Q}_0$ do
        if $\mathsf{fp}_{\mathcal{N}}(p) \leq \tau$ then
            Generate an ordered list of descendants
            $\mathcal{Q} = \nabla_1(p) \cup \cdots \cup \nabla_k(p)$
            $\mathcal{M}_1 =$ MaxFPSafePatterns($\mathcal{Q}, \tau$)
            if $\mathcal{M}_1 = \emptyset$ then $\mathcal{M} = \mathcal{M} \cup \{p\}$ else $\mathcal{M} = \mathcal{M} \cup \mathcal{M}_1$
        end
    end
    return $\mathcal{M}$
end
```

Algorithm 1: Basic algorithm for finding fp-safe patterns.

This condition stated above is rather strong, as it forces us to traverse the same pattern several times. The simplest way to avoid repeated visits is to store all generated patterns and test whether we have already visited that pattern. However, there are more efficient alternatives.

Let us consider a graph of Q-connected patterns where there is an arc from q to p if the pattern q is in its candidate list $Q(p)$. Let F be the filtering operation that keeps only a subset of candidates Q in each step. Then we can define F-connected graph where there is an arc form p to q if the pattern q is in the reduced candidate lists $F(Q(p))$. Now if all patterns reachable in Q-connected graph are reachable in F-connected graph, then applying F to reduce the candidate list in Algorithm 1 does not change the outcome. Indeed, if fp-safe pattern q is reachable form p then it is also reachable through F-reduced candidate lists. Hence, we need to define filters that preserve connectivity while reduce as many alternative paths in the Q-connected graph. As the same argument holds if we start from F-connected patterns and then apply another filter, we can build filter by specifying several sub-filters that eliminate different types of alternative paths.

Basic pattern generalization operators. Now it is relatively easy to see that the following generalization steps assure that all direct descendants of a pattern are generated.

Operator ∇_1: For pattern p, we generalize all of its tokens by minimal amount. More formally, we iterate over all possible tokens p_i and all of its attributes $attr$ and generate a new pattern $q = p_1, \ldots, p_{i-1}, t, p_{i+1}, \ldots, p_n$ where the token t is a copy of p_i with a single attribute $t.attr$ replaced with its parent. The output consists of all generated patterns.

Operator ∇_2: For a pattern p, we generate two patterns $q_1 = p_2, \ldots, p_n$ and $q_2 = p_1, \ldots, p_{n-1}$. We delete q_1 or q_2 if some extraction tokens are missing form the pattern. The output consists of remaining patterns.

Basic filters for candidate patterns. Note that there are only two ways to generalize a pattern: either we omit some tokens form the ends or generalize tokens. Moreover, the generalization can be always carried out so that we first do omission and after that generalization of tokens. Hence, we can filter candidate patterns based on the four simple rules. Let a *current generalization path* be the sequence of generalization steps from a seed pattern to a candidate pattern q made by Algorithm 1. Then the current generalization path must satisfy the following conditions:

F_1: No omission steps are allowed after we have applied a token generalization step.

F_2: No omission steps form the beginning are allowed after we have deleted a token form the end.

F_3: Individual symbols in the pattern must be generalized from beginning to the end.

F_4: Generalization steps on the individual tokens are applied in canonical order.

Proposition 3.2.2. *The filtering steps F_1–F_5 preserve connectivity. For a single seed pattern s the filtering steps assure that a candidate pattern can be visited only once.*

Proof. The rule F_1 does not break connectivity, as if a pattern q is reachable from seed s then it is also reachable if we do omission steps first. The rule

F_2 does not break the connectivity as we can start deleting tokens from the beginning. Similarly, we can generalize tokens in the pattern from left to right. The rule F_4 does not prevent us from generalizing, it just assures that we do generalization steps in unique order. Note that rules F_1–F_4 together fix a canonical generalization path for each pattern and thus no candidate pattern can be visited more than once. □

When the set of seed patterns consists of several patterns then a pattern might be reachable from several patterns and thus visited more than once, although we obey rules F_1–F_4. Such double visits can be caught with the following additional rule:

F_5: Remove q from candidate patterns if q is a generalization of two seed patterns s_i and s_j and the current generalization path does not start from the pattern $s_{\min\{i,j\}}$.

Proposition 3.2.3. *The filtering steps F_1–F_5 preserve connectivity. For any seed set $\mathcal{Q}_0$ the filtering steps assure that a candidate pattern can be visited only once.*

Proof. Let us prove that the rule F_5 alone does not break connectivity. Assume that a pattern q is reachable from many seed patterns and let s_i be the pattern with the minimal index among them. Then the rule F_5 has no effect if the current generalization path starts from s_i. Since rules F_1–F_4 do not break connectivity for individual seed patterns, rules F_1–F_5 together also do not break connectivity. The claim follows, as q can be a valid generalization of a single seed. □

Finding optimal threshold $\tau \in [0, 1]$ for our pattern mining tasks depends on the characteristics of the training corpus. One assumption, we have to make, is that the relations are distributed similarly in the testing corpora, where we plan to apply the patterns and particular machine learning models. Also, the language itself should be similar in both training and testing corpora.

Good seed patterns. To apply Algorithm 1 in practice for mining fp-safe patterns, we need a set of seed patterns. The easiest way to create them from examples is to treat each labeled example sentence as a pattern. We can also fine tune such patterns by manually excluding words that are irrelevant. To get the corresponding pattern, we trim all irrelevant tokens from both ends. If an irrelevant token occurs is in the middle of the remaining pattern then we delete all of its attributes. In more advanced cases, the user can even specify varying number of wild-card patterns in the middle. As such fine tuning is rather tedious, we can also do semi-automatic trimming of patterns by specifying *context radius*, i.e, the maximal number of tokens before and after a the first or last extraction token. These can be specified separately for the beginning and end.

Complexity. It is difficult to provide exact estimates for running time of Algorithm 1. For each pattern of length n, there are $O(n^2)$ different consequent subpatterns i.e possible generalizations by omitting tokens from beginning or end. For each token, there are constant number of generalizations, if we fix the number of attributes and therefore $O(n)$ generalizations for a single subpattern. Given total of N patterns, the worst case complexity of the algorithm is thus $O(n^3N)$. The estimate could be more exact, but we see that it mostly depends on the length and number of seed patterns.

Average complexity depends on how many common generalizations the patterns have, the patters mining threshold and the corpus itself the process is carried out.

In practice, we might expect up to 500 different seed patterns for a relation, although we can always take a sample, if we have more of them. The length of the patterns depends on the arity of the pattern and the the size of context radius. For unary patterns of context radius of two, average pattern length is about five or six tokens. For binary patterns, this may be up to 10 or more. The constant number of generalizations, as we have twelve attributes and longest path from root to leaf in any of our attribute value trees is four, is $4^{12} = 16777216$. In practice, most tokens use only about four or five attributes, thus the total number of generalizations stays usually less than 100 for a single token. But we still might need to evaluate up to $500 \cdot 10^3 \cdot 100$ patterns for a binary relation in worst case scenario and maybe 5-10% of patterns in an average case, if we consider the threshold and common ancestors of the patterns. This is still infeasible in practice without using additional heuristic steps discussed in Chapter 4.

3.3 Basic algorithm for finding fn-safe patterns

To find maximal fn-safe patterns, we have to specialize a list of seed pattern until the rate of true positives falls below a fixed threshold. As such, the task is completely dual to finding maximal fp-safe patterns. However, there are few aspects to note.

First of all, it is much more difficult to estimate the rate of true positives. Ideally, we would have fully annotated corpus and could easily construct unbiased sample of positive examples $\mathcal{P}$ needed to compute tp. In reality, complete inspection is impractical, as relations do not occur frequently enough in most corpora. Hence, the sample is commonly constructed by manual selection of representative examples or using some a sort of semi-automatic method for condensing the original corpus before doing complete inspection. As a result, the sample does not cover all types of occurrences and the estimate on true positives is likely to be overestimate of $\mathsf{tp}_{\mathcal{C}}(p)$.

Secondly, the search algorithm is very close to frequent sequence mining with few modifications. Frequent sequence mining starts from a single symbol and grows it by adding letters, whereas we can add restrictions to attributes one by one. Hence, the potential depth of a search tree can be much larger.

In our case, we still need to consider the extraction areas of the patterns, which can be of varying width. We cannot start from single symbols, but instead from a group of symbols, where the lengths of the extraction areas are taken from a list of initial seed patterns. Also, when setting thresholds, we must take into consideration that a single binary and any other n-ary ($n \geq 2$) pattern cannot always yield very high recall as examples may contain extraction areas for X and Y in different order, thus making it impossible for some patterns to even match certain sentences.

The algorithm and proof of correctness is analogous to the algorithm for mining *fp-safe* patterns. The main difference is that instead of generalization methods we use specification steps.

Basic pattern specification operations. The specification steps are opposite of operators Δ_1 and Δ_2. For finding fp-safe patterns, we generalized the attributes by a minimal step, here we specify them by a minimal step.

Operator Δ_1: For pattern p, we specify all of its tokens by minimal amount. More formally, we iterate over all possible tokens p_i and all of its attributes $attr$ and for each token and direct child of the attribute generate a new pattern $q = p_1, \ldots, p_{i-1}, t, p_{i+1}, \ldots, p_n$ where the token t is a copy of p_i, such that $\mathsf{parent}(t.attr) = p_i.attr$. The output consists of all generated patterns.

Operator Δ_2: For a pattern p, we generate two sets of patterns $\mathcal{Q}_1 = \{t, p_{1:n} : \mathrm{cover}(t, p_{1:n}) \neq \emptyset\}$ and $\mathcal{Q}_2 = \{p_{1:n}, t : \mathrm{cover}(p_{1:n}, t) \neq \emptyset\}$, where the attribute values of t are leaves in their respective attribute value trees. The set $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ consist of all generated patterns.

Basic filters for candidate patterns. As with generalization, we can also specify here the order of specifications, such that for a single seed pattern p, we do not generate duplicate specifications.

F_1: No extension steps are allowed after we have applied a token specification step.

F_2: No extension steps form the beginning are allowed after we have extended a token form the end.

F_3: Individual symbols in the pattern must be specified from beginning to the end.

F_4: Specification steps on the individual tokens are applied in canonical order.

F_5: Remove q from candidate patterns if q is a specification of two seed patterns s_i and s_j and the current specification path does not start from the pattern $s_{\min\{i,j\}}$.

Chapter 4

Heuristic search

In this chapter, we discuss an alternative approach to depth first pattern mining technique discussed in Chapter 3. This approach uses similar patterns and their common ancestors to build up a hierarchal tree. The process usually requires less steps than depth first search, but it is not enough to yield maximal patterns. Instead, this approach can be used to generalize seed patterns to their common generalizations and input those to depth search first algorithm for further refinement.

4.1 Pattern alignment

Consider a set of seed patterns $\mathcal{P}$, which are initiated from a number of annotated examples. Patterns encoding the relation of the same word order or similar structure are more likely to have common generalizations with not many generalization steps. These common generalizations cover the examples of the original patterns and additionally cover many other smaller variations as well. Algorithm 1 generates all such common generalizations, if their fp-rate is small enough. But we can also generate such generalizations directly.

Let us denote the lowest common ancestor generalization of pattern p and q as

$$r = \mathsf{LCA}(p, q) \enspace ,$$

where $p,\, q \preceq r$ and there does not exist pattern m, such that $p, q \preceq m$ and $m \prec r$. If the extraction tokens of the patterns p and q cannot be aligned, we define $\mathsf{LCA}(p, q) = \perp$, meaning that it does not exist.

How the tokens of the resulting ancestor pattern are formed, can be easily specified. Let p be the pattern, whose extraction token positions appear earlier in the pattern. Let $\delta \geq 0$ be the offset of the extraction tokens of q related to p. To align the pattern, we need to crop δ tokens from the beginning of pattern q.

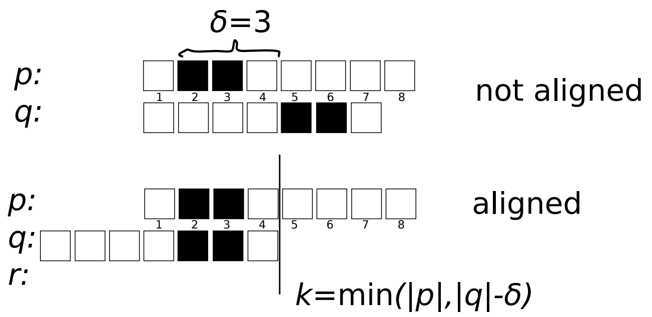


Figure 4.1: Example pattern p and q, where boxes denote tokens and filled boxes the extraction tokens. The offset of extraction token positions of q related to p is denoted as δ.

Let value $k = \min(|p|, |q| - \delta)$ denote the size of the lowest common ancestor pattern r. Therefore, we need to also crop $|p| - k$ tokens from the end of p and $|q| - \delta$ patterns from the end of q. Thus, the lowest common ancestor pattern will be

$$r = \mathsf{LCA}(p, q) = \mathsf{LCA}(p_1, q_\delta), \mathsf{LCA}(p_2, q_{\delta+1}), \ldots, \mathsf{LCA}(p_k, q_{\delta+k}) \ ,$$

if operator LCA with tokens arguments returns the token, where attribute values are lowest common ancestors of respective attribute value trees. See Figure 4.1 for an example. If we used patterns p and q as seed patterns directly for Algorithm 1 and if $\mathsf{LCA}(p, q)$ is frequent, we can omit evaluating all patterns in set

$$\{r \ : \ p \prec r \prec \mathsf{LCA}(p, q) \vee q \prec r \prec \mathsf{LCA}(p, q)\} \ . \tag{4.1}$$

4.1.1 Pattern similarity measures

In general, we can take least common ancestor patterns of any two patterns that align and use Equation (4.1) to reduce the search space, if the pattern is frequent. However, if the generalizations are not frequent, we do unnecessary work. Therefore, we should consider common ancestors of patterns, which are more similar to each other, i.e, the number of required generalization steps to gain the LCA are small. For that purpose, we need a way to measure the similarity between the patterns.

Sentences encoding a particular relation usually have similar language constructs, specific words or other means that makes it possible to recognize the examples. For example, consider sentence

President Obama kohtub president Ilvesega Kadriorus.

Word “kohtuma” (“to meet”) allows to recognize that the subject and the object of the sentences will meet each other. In Estonian, word order is not consistent, although subject-verb-object (SVO) is considered the main order. The above sentence can be also written in VSO order

Kadriorus kohtub president Obama president Ilvesega.

Or we can write the sentence using two subjects as

Kadriorus kohtuvad president Obama ja president Ilves.

These different word orderings cannot be captured by a single pattern and most of the time a single word ordering requires more patterns due to variations in the language as well. But examples having the same word order or sentence structure are more similar to each other than examples with different word ordering.

There are several possible ways we could measure the similarity. For instance, we could use Levenshtein distance, but in terms of our patterns, we cannot insert or delete tokens in the middle, thus this measure would not capture the behavior of the patterns perfectly. Another option is to use Hamming distance. Although the patterns can be of different length, comparing them can be done, if we align them and chop the non-aligning tokens. We define the Hamming distance for pattern p and q as $\mathsf{dist}(p, q) = \infty$, if they cannot be aligned and

$$\mathsf{dist}(p, q) = N + \sum_{i=1}^{k} \mathsf{dist}(p_i, q_{i+\delta}) \tag{4.2}$$

otherwise, where $N = \delta + |p| + |q| - 2k$ is total number of tokens that are chopped off, $k = \min(|p|, |q| - \delta)$ and δ is the alignment offset. The distance measure of single tokens t and u is defined as

$$\mathsf{dist}(t,\, u) = \begin{cases} 0 & t \preceq u \vee t \succeq v \\ 1 & otherwise \end{cases} .$$

The distance measure reflects the number of generalization/specification steps we need to do to convert one patterns to another. This assumes that the patterns can be aligned. Each chop/extend step will contribute value 1 to the distance as do the aligning tokens that are not equal nor cannot be generalized/specified to each other directly. The measure (4.2) also satisfies the triangle inequality, i.e, $\mathsf{dist}(p, q) + \mathsf{dist}(q, r) \geq \mathsf{dist}(p, r)$. Also, it is easy to see that for patterns p, q, r

$$p \preceq q \preceq r \Rightarrow \mathsf{dist}(p,q) \leq \mathsf{dist}(p,r) \tag{4.3}$$

and

$$\begin{aligned} \mathsf{dist}(p, \mathsf{LCA}(p,q)) \leq \mathsf{dist}(p,q) \\ \mathsf{dist}(q, \mathsf{LCA}(p,q)) \leq \mathsf{dist}(p,q) \end{aligned} \quad . \tag{4.4}$$

4.2 Using the similarity measure as heuristic

The distance measure (4.2) can be used as heuristic to decide, what generalization steps should we do to get . If we have two seed patterns p and q that are similar, their lowest common ancestor $\mathsf{LCA}(p,q)$ will cover the examples covered by both p and q.

To use the heuristic, we first need to calculate the pair-wise distances between all seed patterns. We can traverse the pattern pairs by starting with the most similar, take their lowest common ancestors and add it to result, if it satisfies pattern mining criteria, i.e, it is frequent. We can use the resulting patterns as seed patterns to a new iteration and continue, while we gain new patterns. Each next iteration tries to discover frequent common ancestors from the patterns of previous iterations. This process is analogous to depth first search discussed in (3), but uses larger generalization steps.

The resulting patterns are more close to the pattern mining threshold than the initial seed patterns and due to Equation (4.1) help to avoid potentially large number of generalization steps. Using these patterns as seeds to Algorithm 1, we can refine the output further.

Chapter 5

Patterns as features for machine learning

Given the set of fp-safe patterns for a specific relation, we can apply them individually and treat all matches as occurrences of the relation. However, this might lead to suboptimal performance, as the number of false positives starts to cumulate. Moreover, a simultaneous match of some patterns might be much stronger indicator that the extraction is correct than a match of a single pattern. Therefore, we can significantly improve the results by using machine learning for aggregating results. The resulting procedure consists of two phases.

In the first phase, individual patterns are used to match sentences. For clarity, consider unary patterns that extract a single word. Then the pattern matches a word or not and we can encode this as a zero-one feature. In the second phase, these zero-one features are used to decide whether each word is in the relation or not. The latter is a standard classification task. To solve it, we need positive and negative examples. Positive examples can be generated from known occurrence of the relation by computing the corresponding feature vector. Negative examples can be analogously generated from known non-occurrences.

5.1 Feature Extraction

We use the outputs of patterns of the model as features to the classification methods. In particular, we encode the example set as a $n \times m$ matrix M, where n is the number of examples and m the number of patterns. Each example denotes an element $e \in \mathcal{C}$, where

$$\mathcal{C} = \bigcup_{p \in P} \mathrm{cover}(p)$$

is the union of all cover elements of the patterns and particular element $\mathsf{M}_{\mathsf{i,j}} = e_i \in \mathrm{cover}(p_j)$. In case we are building the features for training, we encode the labels as a vector $\mathbf{v}$ of length n, where $\mathbf{v}_\mathsf{i}$ is `True`, if element covers an annotated example and `False` otherwise. The matrix M together with label vector $\mathbf{v}$ can be used as an input to any *supervised learning* classification method to build a model. Note that in case no pattern matches an annotation, we exclude it from the training set as we need a match to discover it in the first place. However, this is not the problem, if the seed patterns originate from the annotations.

5.2 Classification Methods

In this work, we are experimenting with several classification methods and compare their performance: Count classifier, SVM classifier, Random Forests classifier and Naïve Bayes classifier.

Count classifier. We use count classifier as a baseline method in our experiments. It works simply by counting the number of patterns m that match a given example and yield 1, if $m \geq \tau$ and 0 otherwise, where $\tau \in \mathbb{N}$ is the threshold. In our experiments, we use $\tau = 1$, as the resulting classifier reacts whenever at least one of the patterns matches the example, making it the most sensitive classifier. The number of matches m can be directly used as an evidence score. We refer to count classifier, where $\tau = 1$ also as OR-classifier.

SVM classifier. Support vector machine (SVM) is type of classifier, that has high generalization ability [CV95]. In this work, we use linear SVM, although the classifier could also use non-linear transformations to transform the data into higher dimensions for possible better separability. However, in our case we already have many zero-one features, thus using non-linear transformations might not give any advantages. For computations in our tool (see Appendix A), we use `R e1071` package [DHL+05] for linear SVM functionality, which is a interface to `libSVM` library [CL01]. The package allows to use regular two-class classification and can also output probability values for different examples.

Random Forests classifier. Random forests classifier is an ensemble classifier, that uses a combination of tree predictors to vote for the best class. Each individual tree is built by randomly sampling a vector of values from the dataset [Bre01]. We use `R` package `randomForest` [LW02], which is an interface to original program written by Breiman. Similarly to `e1071` package, it can output both resulting class vectors and probabilities for the examples.

Naïve Bayes classifier. Naïve Bayes classifier is a probabilistic classifier that assumes that all the features are independent. Given an example e, the probability that e encodes a relation is given by

$$\Pr[e|f_1, \ldots, f_n] = \frac{\Pr(e)\Pr(f_1|e) \cdot \ldots \cdot \Pr(f_n|e)}{\Pr(f_1, \ldots, f_n)} \quad ,$$

where n is the number of features and feature f_i is true, if pattern i covers the example. An example is classified as positive, if $\Pr[f_1, \ldots, f_n|1] \geq \Pr[f_1, \ldots, f_n|0]$ and false otherwise. The `R e1071` package also implements a Naive Bayes classifier, which we are using in our experiments.

5.3 Case study: named entity recognition

We start our discussion of empirical results from named entity recognition, as unary relations are less ambiguous and the state of the art results are known for the news article benchmark [Tka10]. We first discuss performance for individual relations (person, organization and location) separately and then compare the results. For each experiment, there are many parameters to tune. We can choose the classification method, set a bound on the false positive rate and finally tune the classifiers threshold. In order to test, which of these combinations leads to optimal result, we conducted the following experiment.

The setup. As we could not tell the best choice of parameters beforehand, most specifically the pattern mining FP-rate threshold, we chose to experiment with different values. For each particular threshold $\tau \in [0, 1]$, we ran a 5-fold random cross-validation.

First, we randomly split the corpus into a 80% training and 20% testing sample. Second, using the threshold τ, we mined the FP-safe patterns from the training set. Then, we initialized the models for Count, SVM, Naïve Bayes and Random Forests classifiers using the feature table built from the pattern covers. Each row of the feature table represented a match by any of the patterns. Each element of the row denoted whether a particular pattern matched the element

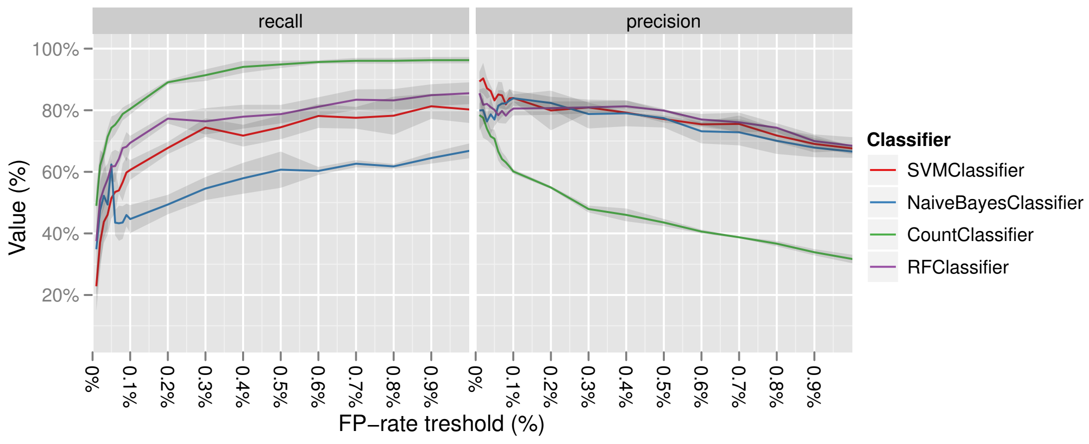


Figure 5.1: Pattern mining FP-rate threshold versus recall and precision for person relation with random 5-fold cross-validation. The shaded regions depict the standard deviation.

or not. We made the feature table balanced, so that there would be equal amount of true positive and true negative matches. Of course, in addition to the feature table, we needed to provide a label vector denoting the true positive and true negative matches.

Next, we matched the patterns on the testing set and measured precision, recall, FP-rate, FN-rate of all the classification models trained previously. Additionally, we also stored the number of patterns mined and calculated the area under curve (AUROC) value for each respective classifier. In this step, we used the default parameters of the classifiers.

Later, we drew receiver operating characteristics (ROC) curves for reasonable FP-rate thresholds by inspecting the area under curve (AUROC) values from the previous steps. Particularly, we chose minimal FP-rate thresholds where AUC values were equal to or near to local maximums.

Important note. The results described in this section were carried out with the version A of the prototype. Please refer to Appendix A for the details.

Annotating persons. As expected, the overall performance largely depends on the fp-threshold used in the pattern mining phase. A low threshold leads to patterns with high precision and low recall. Hence, the precision is initially quite high for all classifiers and then starts to decline. Figure 5.1 clearly shows that machine-learning methods significantly reduce the rate of false-positives. In fact, these methods have roughly the same precision over the entire range.

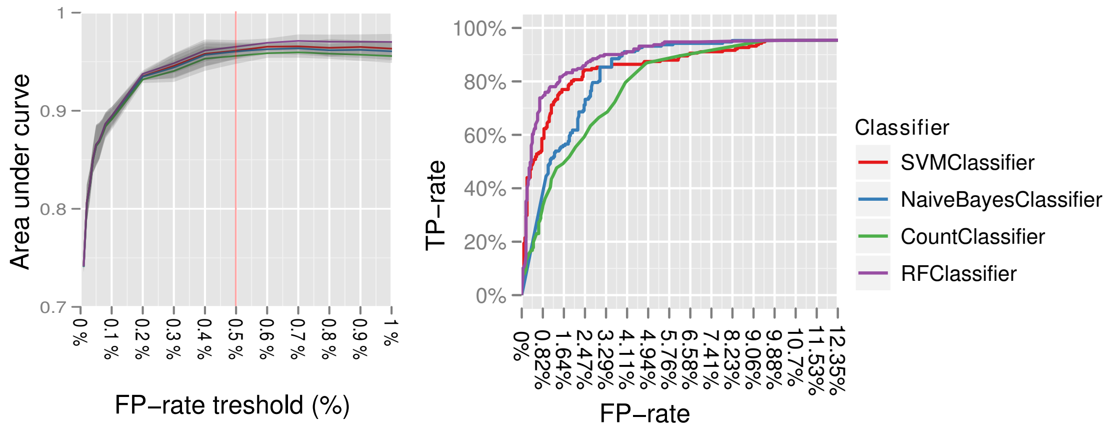


Figure 5.2: The area under curve (AUROC) metric for person relation (left plot) on news corpus. The red line marks the threshold 0.5%, where the ROC curves are plotted in right.

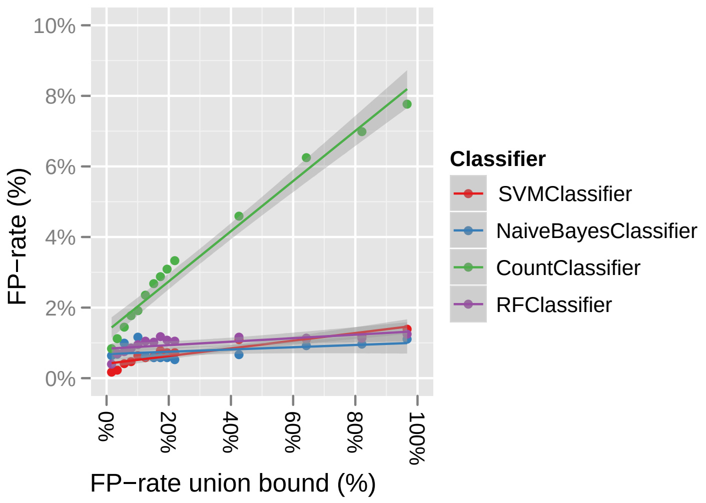


Figure 5.3: Theoretical union bound vs actual union bound of patterns for person relation. Plotted lines depict the linear regression for each classifier.

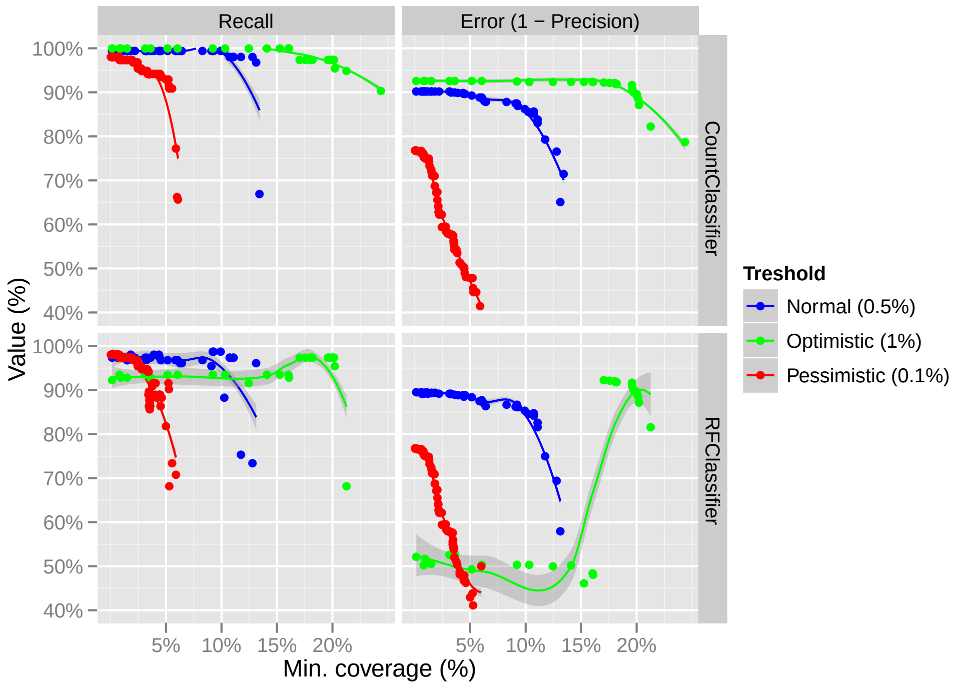


Figure 5.4: Recall and error (1-precision) depending on coverage of patterns of person relation with count and random forests classifier.

The only exception is in the beginning, where the performance of Support Vector Machines is significantly higher attaining 90%. On the other hand, there is a big difference in recall and random forest is clearly the best option. Also, note that the recall of naive Bayes is initially comparable but there is a sharp drop. This is an expected result, as naive Bayes is designed to aggregate independent features. For low thresholds, this assumption is satisfied, while high thresholds introduce correlated patterns.

Recall graph allows us to estimate how many occurrences of person not easily discoverable by patterns. Indeed, note that the recall of the OR-classifier converges to 95% indicating that 5% of true positives do not follow any pattern with reasonable specificity. Figure 5.1 also reveals that it is relatively easy to drop irrelevant sentences so that almost all occurrences of person relation are preserved while occurrence rate is above 50%.

The dependencies depicted in Figure 5.1 are approximate, since they were obtained with default cutoff thresholds. Results can be further optimized by changing the cutoff point. However, the latter leads to two-dimensional optimization task with many objectives. To simplify matters, we first choose pattern-mining threshold based on the AUC scores and then use the corre-

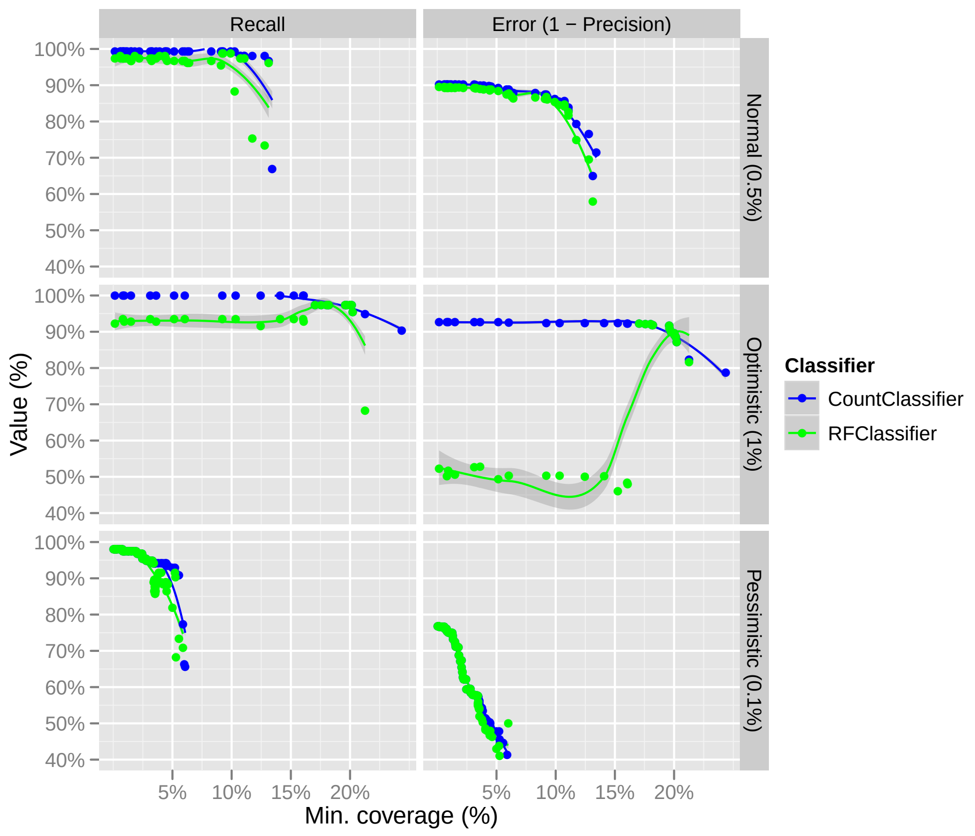


Figure 5.5: Recall and error (1-precision) depending on coverage of patterns of person relation with count and random forests classifier.

sponding ROC curves to characterize performance of individual classification methods. Figure 5.2 depicts the behavior of various classifiers. As one can see random forest and SVM have almost equal performance, though random forest has constantly slightly above.

Classifier	5%	10%	25%	50%
Random Forest	1.5%	**62.3%**	**84.27%**	**94.7%**
Linear SVM	3.1%	48.6%	80.1%	90.5%
Naive Bayes	NA	NA	53.9%	94.2%
Count	**5.2%**	7.3%	33.5%	89.5%

Table 5.1: Recall percentage for various FDR levels of person relation. NA means that a specific method could not provide output with FDR rate that low with patterns mined with FP-rate threshold of 0.5%.

Although the ROC curves are great for comparing different classifiers they do not reveal how useful they are in practice. Hence, we also consider recall under fixed False Discovery Rate (FDR). That is, we consider how many occurrences we can reveal if we allow at most α% of false positives in the final annotation. The standard thresholds to consider are 5, 10, 25 and 50% where the first means one mistake up to nine correct guesses and last one mistake up to correct guess. Table 5.1 depicts the corresponding results. The result clearly shows that patterns contain useful information about person relation but the current results are insufficient for automatic labeling.

Classifiers with default parameters are capable of yielding precision of 70% with recall of 90%, if we mine the patterns with FP-rate threshold 0.01. As a comparison, the NER application used in birthday and event location relations had both precision and recall of 90%, if best features were used [Tka10].

As the patterns are mined from the training set with a fixed FP-rate threshold, theoretically the maximal FP-rate depends also on number of patterns. If each pattern is allowed to output 5% of false positives, then with n patterns, the union bound for the metric is $5n\%$. In Figure 5.3, we have plotted the theoretical FP-rate union against the real FP-rate of all the patterns with person relation. We see that the actual FP-rate and the theoretical union bound are in linear relationship. With OR-classifier, the actual FP-rate is about 8% of the theoretical union bound. This means that the covers of the patterns are not disjoint and many examples are covered by more than one pattern. For machine learning methods, the FP-rate is only about 1.5% of the theoretical union bound, meaning that in fact they yield less false positives than the OR-classifier.

In Figures 5.4 and 5.5, we have plotted minimal coverage of patterns and compared the recall and precision error with pessimistic, normal and optimistic thresholds for person relation. By coverage of a single pattern, we mean the percentage of false positives and true positives the pattern covers. We see that the optimistic threshold really yields more generic patterns that cover more examples. With OR-classifier and few high coverage patterns, we can achieve quite high recall and adding more specific patterns to the set, we cover almost all of the examples. As OR-classifier classifies all examples covered by all patterns as positives, we also increases the error. We see that RF behaves similarly, although it has better precision and slightly lower recall. If we have only few very generic patterns, the error of RF classifier is as high as the error of OR-classifier. But as soon as we add more specific patterns, the RF can produce models with higher precision. From Figure 5.5, we see that with pessimistic threshold and more specific patterns, the machine learning cannot improve the precision much, because the patterns make less errors themselves.

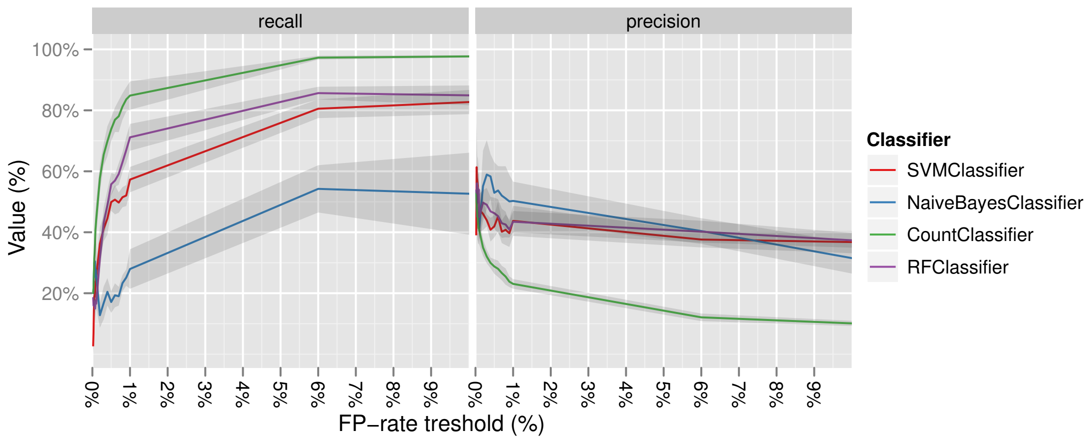


Figure 5.6: Pattern mining FP-rate threshold versus recall and precision for organization relation with random 5-fold cross-validation. The shaded regions depict the standard deviation.

More generic patterns however cover overlapping examples, thus random forest classifier can improve on it.

Annotating organizations. Figure 5.6 shows that pattern-based annotation of organizations is a significantly more challenging task. Again, linear SVM is best for small thresholds, while random forests are best for larger thresholds.

Organizations follow more generic patterns than persons as the recall converges to 98% with OR-classifier and near 85% with RF and SVM with FP-rate threshold about 6%, which is about ten times larger than the threshold for person relation. Filtering of unrelated sentences can reveal us only up to 70% of occurrences while keeping the false positive rate about 50%.

Classifier	5%	10%	25%	50%
Random Forest	0.8%	0.8%	**8.4%**	**61.8%**
Linear SVM	**3.3%**	**3.3%**	6.7%	33%
Naive Bayes	NA	NA	NA	NA
Count	0.8%	0.8%	0.8%	0.8%

Table 5.2: Recall percentage for various FDR levels of organization relation. NA means that a specific method could not provide FDR rate that low with patterns mined with FP-rate threshold of 6%

The FDR values for organization relation are given in Table 5.2, where the fp-rate threshold of the patterns is 6%. We see that we with FDR rate of 50%

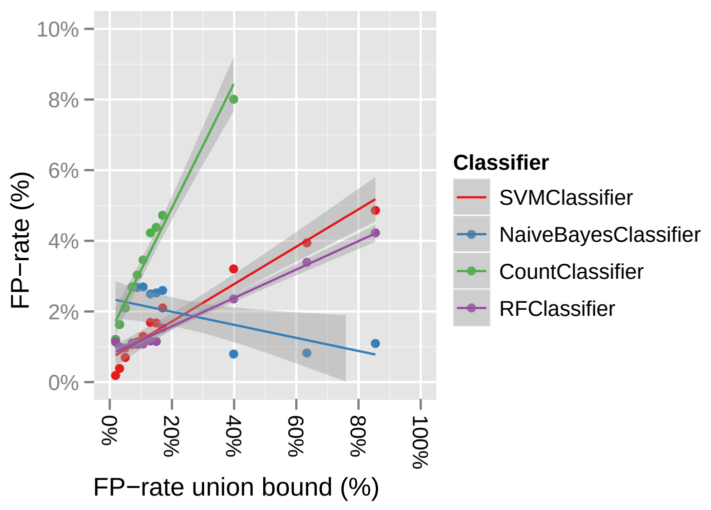


Figure 5.7: Theoretical union bound vs actual union bound of patterns for organization relation. Plotted lines depict the linear regression for each classifier.

we can achieve only recall of 61.8%. With FP-rate threshold of 6%, random forest can achieve precision of 42% with recall of 80%. As a rough comparison, the baseline NER tool has precision of 82% with recall of 76% on the original corpus.

As a result, our approach can achieve feasible recall, but not good enough precision. On the other hand, we did not use lists of know organizations beforehand and made predictions only based on sentence structure.

In Figure 5.3, we see that actual FP-rate is is more close to the theoretical union bound that it was with person relation. For OR-classifier, the actual FP-rate is near to 20% of the theoretical upper bound. With machine learning classifiers other than Naive Bayes, we reach about 5% to 6% of the upper bound. Naive Bayes classifier can decrease the actual FP-rate while the theoretical bound increases. This means that the patterns in organization relation are more independent as it was the case with person relation.

In Figures 5.8 and 5.9, we have plotted the minimal coverage of patterns and their recall with OR and Random Forest classifiers. Compared to person relation, we notice that with normal and optimistic thresholds, there are more generic and independent high-coverage patterns as RF classifier cannot improve the precision due to lack of overlapping in the covers of the patterns. As we add more specific patterns with smaller coverages to the set, the RF classifier will have more information to improve the precision of the model. With pessimistic thresholds, the situation is similar to person relation, although the

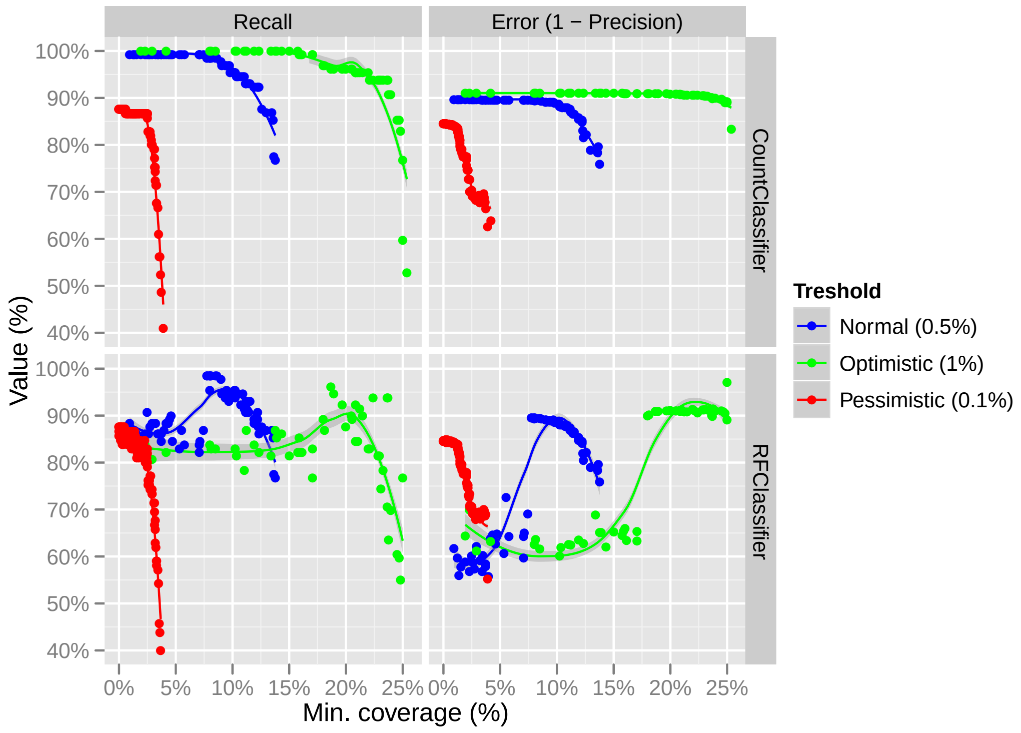


Figure 5.8: Recall and false discovery rate (1-precision) depending on coverage of patterns of organization relation with OR and random forests classifier.

absolute error levels are higher due to more generic patterns in the organization relation in general.

Annotating locations. In Figure 5.10, we see that the performance of annotating locations with pattern-based methods is similar to annotating organizations. Best classifier is again random forest. If we require equal amount of true positives and false positives, we can achieve recall of 62.8% (see Table 5.3) for random forest classifier.

The recall of OR-classifier converges to 98%, which means that the patterns cover most examples in the corpus, but remain too generic. With default settings, the random forest achieves precision of 35% and recall of 87% with FP-rate threshold of 6%. To compare, the baseline NER implementation achieved here precision and recall both of 90% on the original corpus.

The relationship between actual FP-rate and theoretical union bound are similar to organization relation. Similar situation is with the effect on recall and precision depending on the coverage of patterns. Therefore we omit the

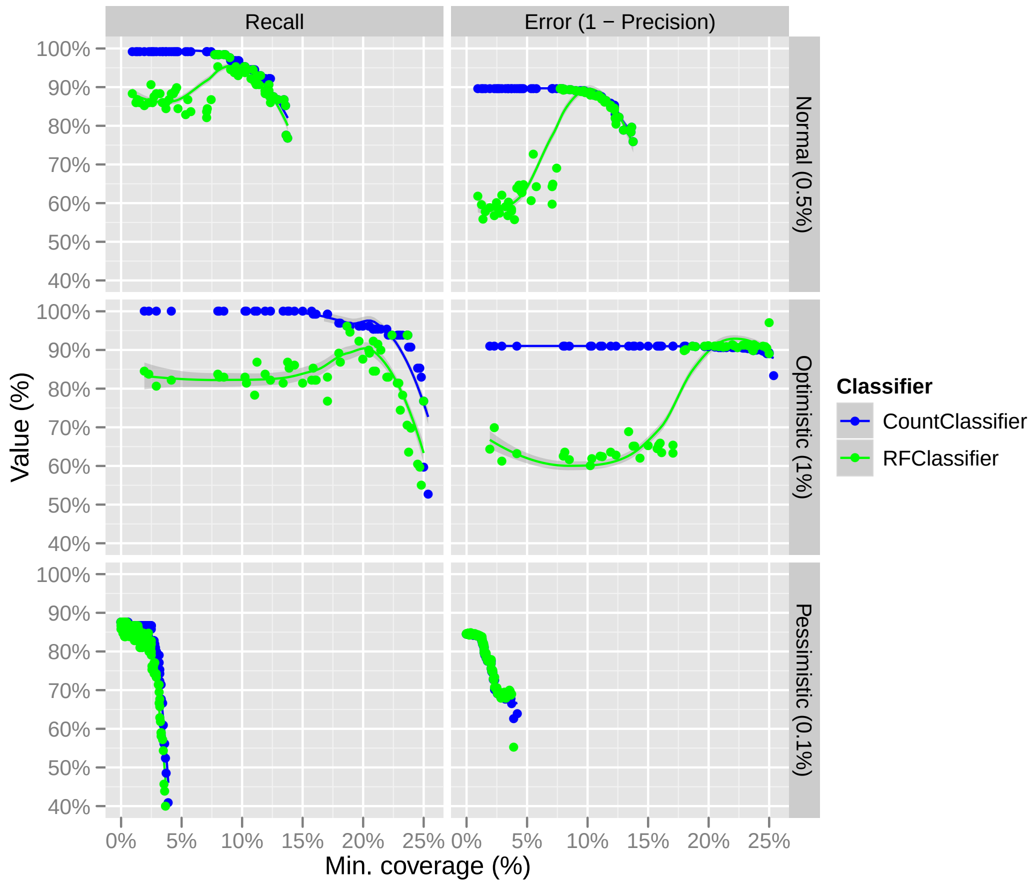


Figure 5.9: Recall and error (1-precision) depending on coverage of patterns of organization relation with OR and random forests classifier.

Classifier	5%	10%	25%	50%
Random Forest	NA	NA	**33%**	**62.8%**
Linear SVM	2.4%	2.4%	13.2%	53.7%
Naive Bayes	NA	NA	NA	17.2%
Count	0.8%	0.8%	0.8%	5.7%

Table 5.3: Recall percentage for various FDR levels of location relation. NA means that a specific method could not provide FDR rate that low with patterns mined with FP-rate threshold of 6%

plots for location relation.

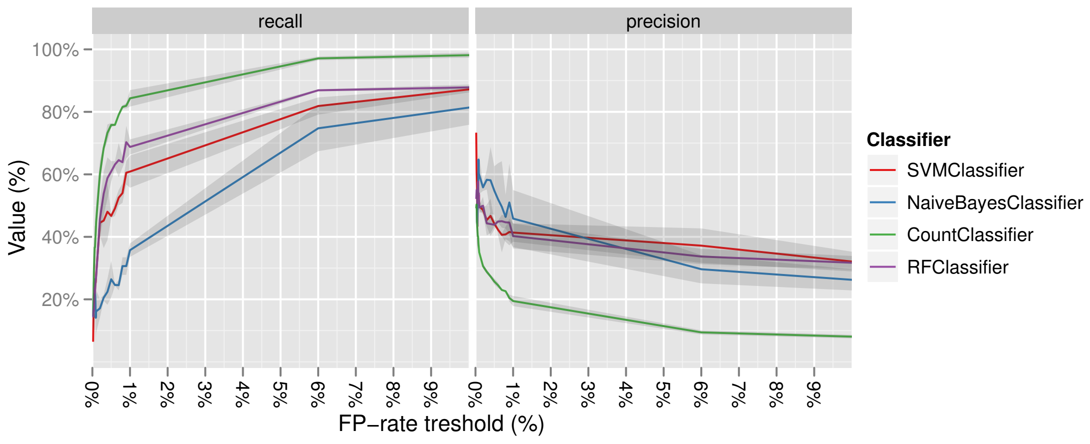


Figure 5.10: Pattern mining FP-rate threshold versus recall and precision for location relation with random 5-fold cross-validation. The shaded regions depict the standard deviation.

Comparison of results. In Table 5.4, we have given comparisons of best recall rates of best classifiers for each tested relation and FDR level. We see that our methods worked best with annotating person names. We could achieve recall of almost 95% with false discovery rate of 50%. With organization and location relations, the recall was only slightly over 61%.

Relation	5%	10%	25%	50%
Persons	**5.2%**	**62.3%**	**84.27%**	**94.7%**
Organizations	3.3%	3.3%	8.4%	61.8%
Locations	2.4%	2.4%	33%	62.8%

Table 5.4: Best recall percentage for various FDR levels for person , organization and location relations of best classifiers.

Our methods are currently not yet feasible for directly annotating named entities. We can achieve acceptable recall, but not good enough precision. This indicates that the examples follow specific patterns, but which are in some cases too generic to achieve high precision. In case of organization and location relations, it might have been the case we had too few negative examples in the corpus, thus under-estimating the FP-rate of the patterns in training phase.

The relationship between the actual FP-rate levels and theoretical upper bounds depend largely on the type of relation. If the relation is simper and we have enough examples, the covers of the patterns have much more overlaps. This means that we agree on the general structure of the sentences, but use

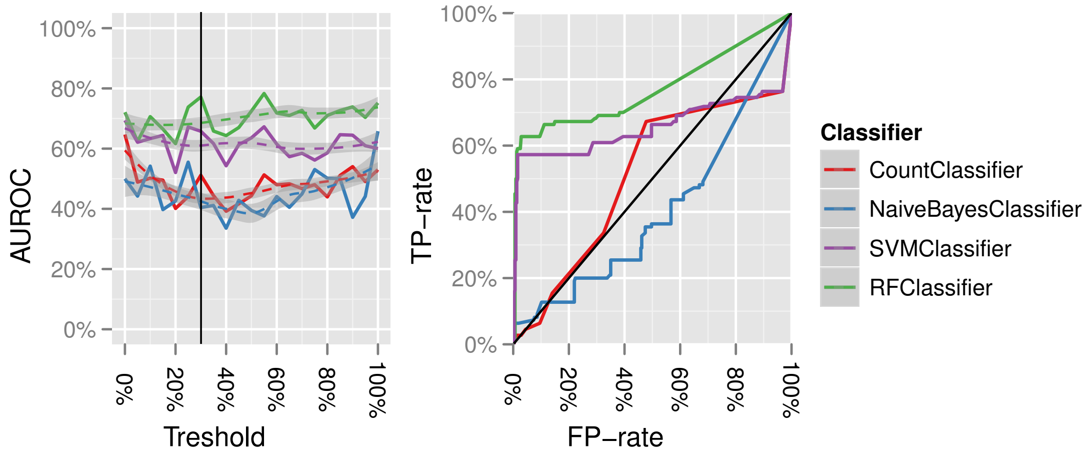


Figure 5.11: Area under ROC curve (AUROC) for binary birthday relation in Wikipedia corpus for different *fp*-rate threshold for pattern mining (left) for 10-fold random crossvalidation. Dashed lines with shaded areas denote smoothed condition mean. In the right, there are ROC curves for different classifiers, where pattern mining threshold is 30% (local maxima).

minor modifications to capture other nuances as seems to be the case with person relation. With organization and location relations, the actual FP-rate was higher and thus closer to theoretical upper bound. The relations are more diverse in the corpus and cannot be captured with the patterns very easily. One thing common to all relations was that machine learning methods definitely improve the precision of the model.

If we look only at high-coverage patterns, we see that we could achieve quite high recall using only a small fraction of optimistic patterns, keeping the error proportionally smaller. This means that the relations can be captured by a small number of very generic high-coverage patterns and further be specified by adding more patterns to the relationship. This, in general, was the case with all three relations.

5.4 Case study: binary birthday relation

In this section, we try the machine learning methods briefly discussed earlier with birthday relation on Wikipedia corpus. We carried out a 10-fold random crossvalidation test, where we mined fp-safe patterns with varying thresholds from zero to one hundred percent. We applied the trained patterns and models on the testing sets and compared various statistical measures. As seed

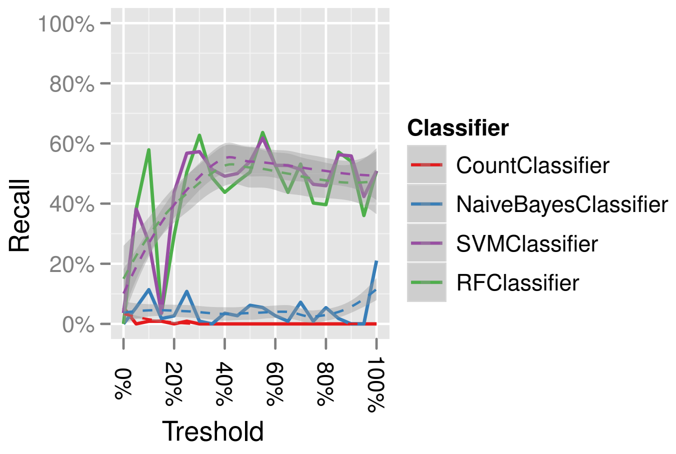


Figure 5.12: Maximal recall for fixed false discovery rate (FDR) of 5%.

patterns, we used annotations in training sets and applied a context radius of two tokens. We used heuristic patterns mining algorithm .

Important note. The results described in this section were carried out with the version B of the prototype. Please refer to Appendix A for the details.

Discussion. Figure 5.11 depicts the area under ROC curve and ROC curves for fp-rate threshold of 30%. We see that random forest (RF) classifier performed best throughout the test, obtaining almost AUROC of 0.8 with some thresholds. The average performance of the classifier tends to increase slightly, when the threshold increases, although it is pretty stable. Similarly, support vector machine (SVM) classifier is stable, but its performance is not as good as of RF classifier. Naïve Bayes and Count classifier perform better with either low or high fp-rate threshold values, but both performed worse than RF and SVM classifiers.

In Figure 5.12, we have depicted the maximal recall for a fixed false discovery rate (FDR) of 5%. This means the recall such that for 19 true positives we have 1 false positive. RF and SVM classifiers can provide recall around 50% with pattern mining threshold of more than 40%. As it is lower with smaller thresholds, this indicates that the covers of abstracted patterns have significant overlaps, such that it is possible to precisely determine true positive matches. The only downside is relatively small recall. Maximal recall for Count classifier is practically always zero, except for very low pattern mining *fp*-rate thresholds. Similarly, this is true for Naïve Bayes classifier. Both these results are expected, as both introduce a lot of false positives. Count classifier classifies an examples as positive, if it has at least one match. Naïve Bayes assumes the patterns are independent (no overlaps in covers), which is not true in most cases.

Chapter 6

Active learning techniques

The main problem of creating a good training set that cover many good positive instances containing the relation is the time-consuming manual process of finding and annotating positive examples. Typical corpora contain many sentences, where only a small percentage may have the relation we are interested in. Therefore, in this section, we are going to describe techniques that lower the amount of necessary manual work, while increasing the quality of the training set. First goal is to reduce the size of the corpus by eliminating true negative sentences. Second goal is to build an corpus of examples covering most relevant examples of the relation we are interested in.

The overall active learning loop is given in Figure 6.1. The initial step is to annotate a small subset of the corpus, where we can deduce initial patterns to begin with. Alternatively, we could also define initial patterns ourselves, but this requires expert knowledge in Estonian language and may not adequate for the specifics of the corpus. For example, medical texts or forum posts frequently contain language that is not typical nor grammatically correct Estonian. Thus, it is safer to get the seed patterns from the annotations. However, the relative scarcity of good examples often forces us to use heuristic patterns to filter out a set of putative examples that must be manually checked afterwards. For instance, in order to discover examples of 'is synonym' relations, we can first filter out sentences that contain synonym pairs and then verify which of them indeed encode the relation. Also, note that the initial annotations should be heterogeneous and cover the most basic examples for different constructs. This is crucial for binary and ternary relations, where extraction tokens can be in different order.

As the second step, we fix a false positive rate limit and mine patterns, which maximize recall. By matching these patterns in the corpus, we can get a new set of predicted examples. Although this set is likely to contain many false positives, the percentage of true occurrences is much higher than in the original corpus. Moreover, we can seek for informative examples by omitting examples we have already learned, i.e., examples that are matched with high precision patterns. As a result, the amount of required manual effort can drop significantly, although it can be still infeasible to look through the entire set of predicted examples. In the latter case, we just review part of it and repeat the cycle until the resulting gains in precision and recall are insignificant.

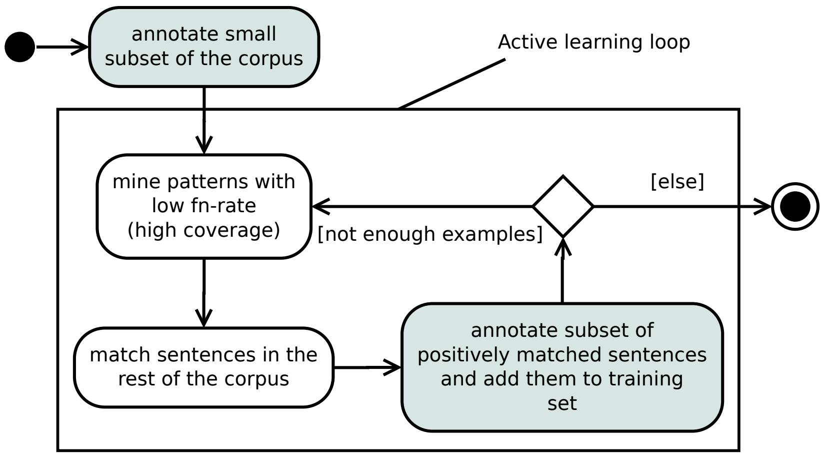

annotate small subset of the corpus
Active learning loop
mine patterns with low fn-rate (high coverage)
[else]
[not enough examples]
match sentences in the rest of the corpus
annotate subset of positively matched sentences and add them to training set

Figure 6.1: Active learning loop. Shaded activities represent the steps that have to be done manually.

6.1 Strategies For Finding New Examples

Given a data set, we can either try to find patterns that maximize precision or recall. The first type extraction mechanism is good as a final fact extraction tool, whereas the other is good for throwing away irrelevant sentences. Hence, in each active learning cycle we train two classifiers: HP-classifier and HR-classifier. Now given a new set of unlabeled sentences, we get three types of matches as depicted in Figure 6.1. The sentences form the intersection $\mathcal{HR} \cap \mathcal{HP}$ are not very informative new examples, since both classifiers agree on them. Sentences from $\mathcal{N}_* = \mathcal{HP} \backslash \mathcal{HR}$ are candidates for negative examples, since they do not fit the general pattern of sentences capturing the relation. Sentences from $\mathcal{P}_* = \mathcal{HR} \setminus \mathcal{HP}$ are good candidates for positive examples as they contain plausible matches not captured by the HP-classifier.

Since the manual verification of new examples is costly, the exact prioritization of new example sentences is an important sub-task. There are three basic strategies:

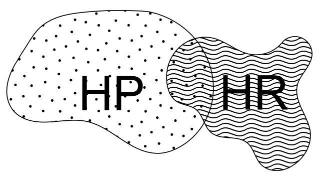


Figure 6.2: Venn diagrams of covers of high recall (HR) and high precision (HP) patterns.

Prioritization based on current behavior of classifiers. The first strategy splits the sets $\mathcal{P}_*$ and $\mathcal{N}_*$ further based on the internal behavior of classifiers. In must cases, HP- and HR-classifier consolidate the results of many patterns by assigning a likelihood score for each potential match. Hence, it is easy to detect candidate sentences of $\mathcal{P}_*$ that are matched by some patterns but rejected due low consensus and totally unmatched patterns. Including the former as an example is likely to improve existing rules, whereas the latter might reveal new patterns. On the other hand, the second type of sentence is also more likely to be false positive. Hence, one needs to balance correctness and informality.

Prioritization based on potential future improvements. The second strategy tries to cluster putative examples based on the patterns they induce. In brief, we mine patterns which cover small but still significant part of $\mathcal{P}_*$ and still have a small false negative rate on the entire text corpus. Now if we partition $\mathcal{P}_*$ according to these new rules, then we can do more intelligent prioritization of examples. Namely, if a putative example is correct then others form the same partition are also more likely to be correct. Similarly, if several sentences from the same cluster are incorrect, then the cluster is likely to contain false positives only.

Prioritization based on extracted items. The third option is to use background knowledge for prioritizing examples. Some words are more likely to be in the relations than the others. As a simplest solution, we can include all sentences that correspond to known relation, i.e., relations that are already revealed by previous examples or given through auxiliary data. Next, we can count the occurrences of a specific relation in the set of putative examples and prioritize sentences based on their over-representation. For that we must fix a null model that adequately describes background noise.

Of course, utility of each method depends on exact implementation details. We need to consider two baseline methods for creating set of positive examples: random sampling from sentences $\mathcal{P}_* = \mathcal{HR} \setminus \mathcal{HP}$ and $\mathcal{HR}$.

6.2 Stopping Criterion

It is important to know when we have enough examples in the training set. Two basic characteristics to consider are increase in recall and increase in precision. Namely, let $\mathcal{E}_i$ and $\mathcal{E}_{i+1}$ be the positive examples used in the i-th and $(i+1)$-st iteration of active learning loop. Let $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ be the corresponding high-precision patterns. Then we can consider changes in the recall and precision of the corresponding HP-classifier:

$$\begin{aligned}\Delta r_{i+1} &= \mathsf{recall}(\mathcal{P}_{i+1}) - \mathsf{recall}(\mathcal{P}_i)\enspace,\\ \Delta p_{i+1} &= \mathsf{precision}(\mathcal{P}_{i+1}) - \mathsf{precision}(\mathcal{P}_i)\enspace.\end{aligned}$$

A change $\Delta r_i \approx 0$ indicates that either we have captured all patterns with reasonable support in the corpus or all example generation procedure does not provides redundant examples. The first situation is most likely to occur if additionally $\Delta p_i \leq 0$. It is natural to assume that $\Delta r_i \to 0$ in the process of adding new learning examples to the set. Thus, if $\Delta r \leq \tau$, where τ is some reasonable threshold, we can stop adding new examples. If we never meet this condition, i.e, the $\Delta r_i > \tau$ and we have run out of positive sentences, then the corpus is most likely too small.

The second set of important indicators is the recall and precision of individual patterns. First of all, if the recall of newly discovered patterns is low, then we have harvested all general patterns and the reminder consists of very specific cases that cannot be distinguished from badly worded sentences. As the relative frequency of patterns depends on the type of text, the latter might indicate that we should switch training corpus for finding more examples.

Finally, note that direct estimation of recall cannot be done without manual examination of all sentences. Since the gradual extension of positive examples during active learning is incomplete and we do not know the count of true positive, we can only estimate the relative change in the recall. However, the latter is sufficient for keeping track of Δr_i.

6.3 Utilization of Negative Examples

Each active learning cycle reveals examples that are known to be true negatives. The distribution of these examples is skewed, as the main aim during the active learning cycle is to get heterogeneous set of positive examples and not uniform sampling over true negatives. Thus, we cannot include them directly to the set of negative examples nor can we use them to estimate ratio of false positives as skewed sampling can bias the estimate.

As baseline strategies sample sentences uniformly form the sets $\mathcal{HR}$ and $\mathcal{P}_*$, we can estimate the relevance of false positive scores computed over the sample. Let α be the fraction of matches of a pattern p that are covered by $\mathcal{HR}$:

$$\alpha = \frac{|\mathrm{cover}(p) \cap \mathcal{HR}|}{|\mathrm{cover}(p)|} \enspace .$$

and let $\mathsf{fp}_\mathcal{S}(p)$ be the rate of false positives computed over the sample $\mathcal{S}$. Then the true rate of false positives $\mathsf{fp}(p)$ must lie in the interval

$$\alpha \cdot \mathsf{fp}_\mathcal{S}(p) \lesssim \mathsf{fp}(p) \lesssim 1 - \alpha + \alpha \cdot \mathsf{fp}_\mathcal{S}(p)$$

where the precision of the approximation is determined by size of the sample $\mathcal{S}$. The upper estimate is uninformative for high precision patterns with $\mathsf{fp}(p) < 1$. Thus, we can order patterns based on the lower estimate and replace non-specific with less general ones.

Another way to increase the precision is based on fixed set of negatives. As usual, we might start form the random sample from the entire corpus. Given predictions from the classifier, we can detect and mark true positives. Now each pattern we can ask whether the false positive rate decreases significantly or not if we omit the pattern and retrain the classifier. By ordering the patterns based on the significance score, we can find patterns that must be made more specific. Also, we can lower the tolerated rate of false positives and see whether the resulting classifier has significantly higher precision. Again, one should stop tweaking if change Δp is insignificant.

6.4 Case study: active learning

We decided to test the practical benefits of the active learning loop described previously. We executed the process given in Figure 6.1. The main difference was that the manual steps were done automatically to provide better overview of the results. Also, we did only one iteration of the active learning loop.

We used birthday and event location relations for the benchmark. We carried out the processing on specially constructed corpus samples, where 20% of the sentences contained the relation. The iteration divided the corpus into three parts: the *initial* learning corpus $\mathcal{I}$, the *additional* corpus $\mathcal{A}$ and the *testing* corpus $\mathcal{T}$, such that $\mathcal{I} \bigcap \mathcal{A} \bigcap \mathcal{T} = \emptyset$.

We mined patterns $\mathcal{P}_0$ with high recall from the initial corpus $\mathcal{I}$. Then, we matched these patterns in corpus $\mathcal{A}$ yielding a cover of extraction tokens from a subset corpus $\mathcal{A}$. We added the true positive matches with particular sentences $\mathcal{C} \subseteq \mathcal{A}$ to initial corpus, yielding an improved initial corpus $\mathcal{I} \bigcup \mathcal{C}$, where $\mathcal{C}$ are the sentences from additional corpus. This step simulated the manual process of reviewing the matches in additional corpus.

To measure the improvement, we mined high precision patterns $\mathcal{P}_1$ from corpus $\mathcal{I}$ and patterns $\mathcal{P}_2$ from $\mathcal{I} \bigcup \mathcal{C}$ and tested them on corpus $\mathcal{T}$. As a result, we calculated improvement rates $\alpha = \mathsf{precision}(\mathcal{P}_2)/\mathsf{precision}(\mathcal{P}_1)$ and $\beta = \mathsf{recall}(\mathcal{P}_2)/\mathsf{recall}(\mathcal{P}_1)$.

We repeated the test with various proportions of initial and additional corpus. The size of the testing corpus was 10% of the original corpus. The proportions for initial, additional and testing sentences were from 10%-80%-10% to 80%-10%-10%.

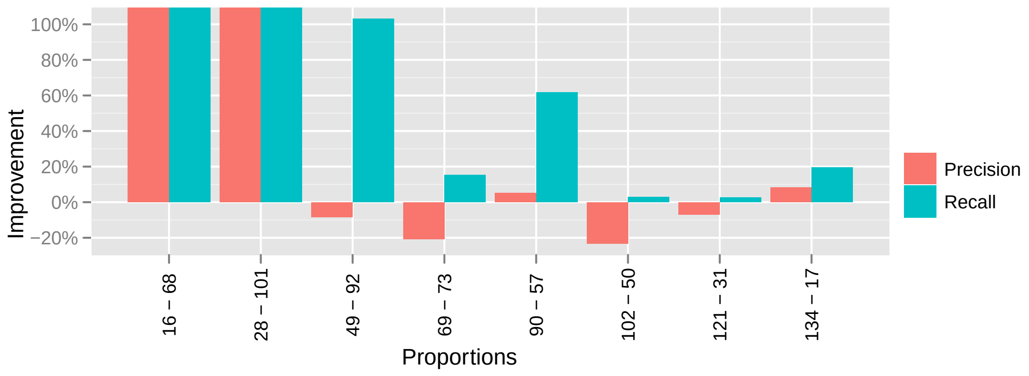


Figure 6.3: The improvement of precision and recall for location relation on Twitter corpus. The proportions define the number of examples in the training and additional corpus.

In Figures 6.3 and 6.4 we see respectively the improvement of precision and recall with location relation on example and sentence level. With example level, we see that active learning loop improves both precision and recall marginally, if the additional corpus is about five times larger than the initial training corpus. This is expected. With few FN-safe patterns we find many new examples, that after annotation give much better overview of the relation, thus improving recall. However, in small initial corpus, it is easier to under-estimate the FP-rate of the patterns due to possible lack of negative examples. Increasing the size of the corpus five times gives more realistic estimates, thus on the testing corpus we achieve better precision.

As the initial training corpus increases, the improvement of recall starts to decline. Also, when the initial corpus reaches the size of half the additional corpus, the precision stays the same or is even little bit worse. Decline in

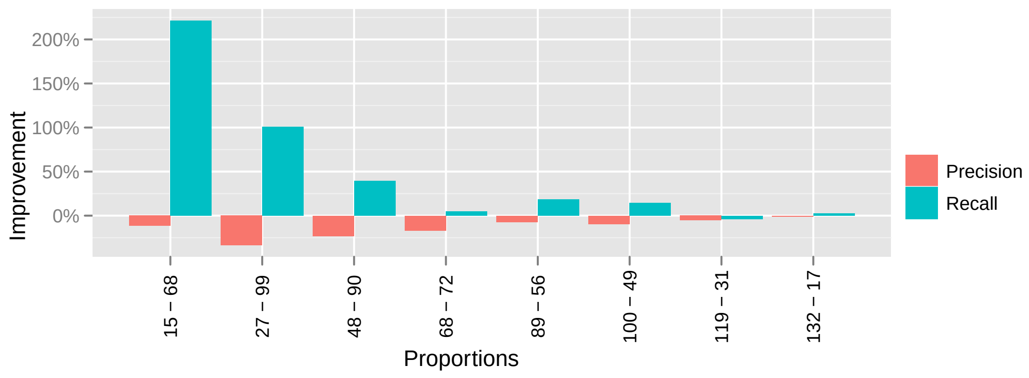


Figure 6.4: The improvement of precision and recall for location relation on Twitter corpus. The proportions define the number of sentences in the training and additional corpus.

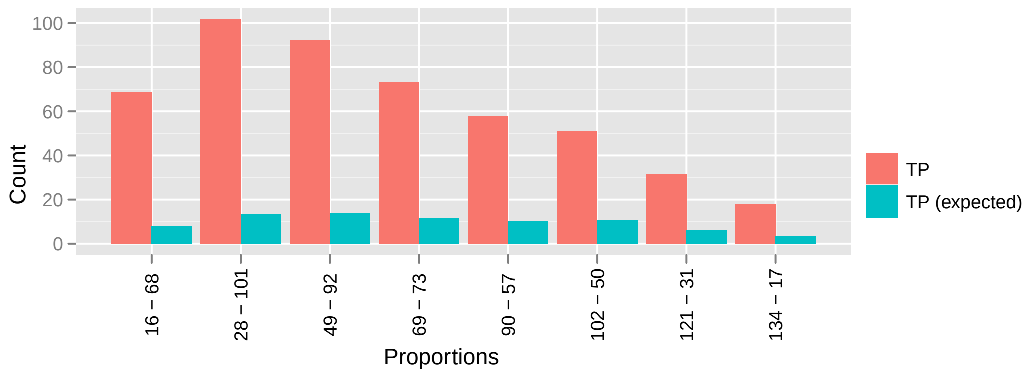


Figure 6.5: Number of true positive examples discovered in Twitter corpus for location relation by FN-safe patterns vs average number of true positive examples discovered from a random sample of size equal to total cover of the patterns.

recall is expected as with larger initial corpus we already cover larger part of the relation.

If the improvement in recall is positive, but improvement of precision is negative, then the patterns we train from newly discovered examples, have less overlapping cover with the existing patterns, but still the same FP-rate threshold, thus increasing the actual FP-rate of the OR-classifier towards theoretical union bound. Resulting in increased recall and decreased precision.

In cases, where the recall does not improve, but precision declines, we have also added many useless examples from the additional corpus that does

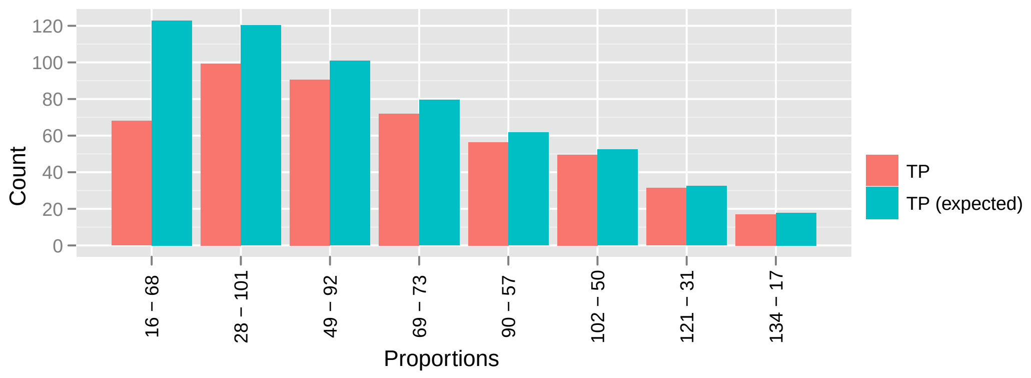


Figure 6.6: Number of true positive sentences discovered in Twitter corpus for location relation by FN-safe patterns vs average number of true positive sentences discovered from a random sample of size equal to total cover of the patterns.

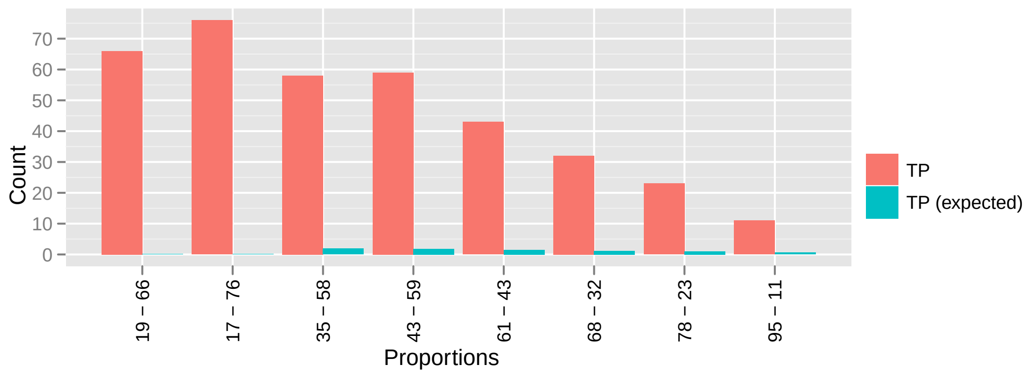


Figure 6.7: Number of true positive examples discovered in Wikipedia corpus for birthday relation by FN-safe patterns vs average number of true positive examples discovered from a random sample of size equal to total cover of the patterns.

not teach us anything new. The problem is that several copies of a similar examples can misleadingly balance the FP-rate of certain patterns during pattern mining by allowing larger number of incorrectly matched examples in the cover, making the pattern more generic. Thus, possibly resulting in inferior performance on the testing set. As can be seen from the Figures, with larger initial corpus the chance to bias the FP-rate of patterns is smaller.

With sentence level, the location relation behaves similarly, but is in fact more stable. Also, there is no sharp increase in precision if the initial corpus is

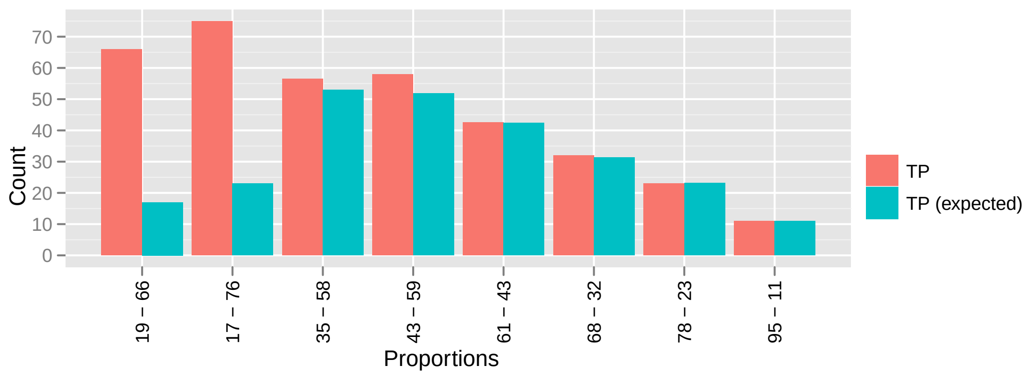


Figure 6.8: Number of true positive sentences discovered in Wikipedia corpus for birthday relation by FN-safe patterns vs average number of true positive sentences discovered from a random sample of size equal to total cover of the patterns.

marginally smaller than the additional corpus. In fact, there is a slight decrease in precision in general. With few FN-safe patterns we can discover many new examples and sentences, thus increasing the recall. Also, a small mismatch at the example level is a correct match at the sentence level. For the reasons similar to example level, by adding more diverse sentences to training process, we end up increasing recall and slightly decreasing precision.

In Figure 6.5, we see that the active learning loop indeed discovered more true positive examples that would have been possible compared to a random sample of examples of the same size. This means, that on example level, active learning can effectively help to find true positive sentences in unannotated texts. However, in sentence level this is not the case as seen in Figure 6.6. The random sampling of sentences would be in fact more efficient way to discover the examples. That being said, the precision of potential new examples can be vastly improved by machine learning methods as in this experiment we use only OR-classifier as the base-line method.

For birthday relation, the results were in general the same, although less stable in terms of improvement of precision and recall. In terms of new covered examples, the active learning methods are again effective on the example level. On the sentences level, the active learning does not increase the number of discovered true positives, except in cases the initial corpus is about five times smaller as seen in Figures 6.7 and 6.8.

Summary

In this work, we formulated fact extraction task for Estonian language using pattern matching based techniques. We developed an Apriori based algorithm for mining the patterns that recognize fragments of sentences and use machine learning algorithms to classify the examples as positives or negatives.

The theoretical part of the work mostly concentrated on pattern matching and pattern mining using the monotonicity property of the patterns, their generalizations and specifications. Although we briefly discussed ideas for active learning and combining the methods with machine learning, the current theoretical results are just starting points for future challenges.

Information retrieval is not an easy task and in case for Estonian language, we had to start the trial and error process from very beginning to get the idea, what kind of approaches work well. The pattern based approach has been very successful for English language, which was the main motivation to start working with patterns. The ideas published by numerous authors working on same problems for other languages have been a great pool of valuable information and inspiration. In the same time, we wished to incorporate as much current language processing machinery available for Estonian as possible such as named entity recognition and syntactical analysis of sentences.

There is a lot of more work to do in the theoretical part: formalize and test different bootstrapping strategies; how to better utilize the information of negative examples in the active learning process; what are the best ways to eliminate statistically insignificant patterns; how to use the approach to improve the output of other language processing tools like NER itself; how to incorporate information from different knowledge bases and ontologies to the process etc. Currently, we tried to cover at least some of the most prominent and hot topics in the pattern based fact extraction literature such as active learning itself.

While the theoretical aspects of any work are important, they need to also be tested in practice. In this work, we ended up implementing two different prototypes capable of carrying out the testing, but with different characteristics and implementation details. We did this to learn, which features are important and should be developed further and what are the main flaws of the prototypes. The implementation strategy was similar to strategy of theoretical work: try to explore as much as it is possible using the time and resources available.

The case studies we carried out included simple model building and its performance evaluation on testing corpora. We saw that our approaches work, but are not very good for thorough “fact extraction”. As the case studies showed, the methods can be precise, but with not very good recall or vice versa. Similarly, the case study of active learning showed several benefits, but definitely there is a lot of room for improvement.

To conclude the work, we now have a very basic overview how our pattern based methods work for fact extraction in Estonian language. The accumulated experience and knowledge is definitely a helpful asset to carry on the research in right direction in the future.

Bibliography

[ABK$^+$07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives, *Dbpedia: A nucleus for a web of open data*, The Semantic Web (Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, eds.), Lecture Notes in Computer Science, vol. 4825, Springer Berlin / Heidelberg, 2007, pp. 722–735.

[AG00] Eugene Agichtein and Luis Gravano, *Snowball: extracting relations from large plain-text collections*, DL '00 Proceedings of the fifth ACM conference on Digital libraries, ACM New York, NY, USA, 2000, pp. 85 – 94.

[BCD01] Lawrence Brown, Tony Cai, and Anirban DasGupta, *Interval estimation for a binomial proportion. with comments and rejoinder by the authors.*, Statistical Science **16** (2001), 101–133.

[BM07] Razvan Bunescu and Raymond Mooney, *Extracting relations from text: From word sequences to dependency paths*, Natural Language Processing and Text Mining (Anne Kao and Stephen R. Poteet, eds.), Springer London, 2007, pp. 29–44.

[BOS04] Paul Buitelaar, Daniel Olejnik, and Michael Sintek, *A protégé plug-in for ontology extraction from text based on linguistic analysis*, The Semantic Web: Research and Applications (Christoph Bussler, John Davies, Dieter Fensel, and Rudi Studer, eds.), Lecture Notes in Computer Science, vol. 3053, Springer Berlin / Heidelberg, 2004, pp. 31–44.

[Bre01] Leo Breiman, *Random forests*, Machine Learning **45** (2001), 5–32, 10.1023/A:1010933404324.

[Bri99] Sergey Brin, *Extracting patterns and relations from the world wide web*, The World Wide Web and Databases (Paolo Atzeni, Alberto Mendelzon, and Giansalvatore Mecca, eds.), Lecture Notes in Computer Science, vol. 1590, Springer Berlin / Heidelberg, 1999, pp. 172–183.

[CBHM10] Andrew Carlson, Justin Betteridge, Richard C. WanEstevam R. Hruschka, and Tom M. Mitchell, *Coupled semi-supervised learning for information extraction*, Proceedings of the third ACM international conference on Web search and data mining (New Yoek, USA), ACM New York, 2010, pp. 101–110.

[CL01] CC Chang and CJ Lin, *Libsvm: a library for support vector machines, software (2001)*, 2001.

[CV95] Corinna Cortes and Vladimir Vapnik, *Support-vector networks*, Machine Learning **20** (1995), 273–297, 10.1007/BF00994018.

[CV05] Philipp Cimiano and Johanna Völker, *Text2onto*, Natural Language Processing and Information Systems (Andrés Montoyo, Rafael Munoz, and Elisabeth Métais, eds.), Lecture Notes in Computer Science, vol. 3513, Springer Berlin / Heidelberg, 2005, pp. 257–271.

[DHL$^+$05] E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, and A. Weingessel, *The e1071 package, 2005*, Software available at< http://cran. r-project. org/src/contrib/Descriptions/e1071. html (2005).

[ECD$^+$04] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates, *Web-scale information extraction in knowitall: (preliminary results)*, Proceedings of the 13th international conference on World Wide Web (New York, NY, USA), WWW '04, ACM, 2004, pp. 100–110.

[EKM$^+$93] Mati Erelt, Reet Kasik, Helle Metslang, Henno Rajandi, Kristiina Ross, Henn Saari, Silvi Vare, and Kaja Tael, *Eesti keele grammatika ii: süntaks*, Eesti TA Eesti Keele Instituut, 1993.

[EKM$^+$95] Mati Erelt, Reet Kasik, Helle Metslang, Henno Rajandi, Kristiina Ross, Henn Saari, Kaja Tael, and Silvi Vare, *Eesti keele grammatika i: morfoloogia, sõnamoodustus*, Eesti TA Eesti Keele Instituut, 1995.

[Goe03] Bart Goethals, *Survey on frequent pattern mining*, Tech. report, 2003.

[GY09] Nikesh Garera and David Yarowsky, *Structural, transitive and latent models for biographic fact extraction*, EACL '09 Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (Stroudsburg, PA, USA), Association for Computational Linguistics, 2009, pp. 300–308.

[HCXY07] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan, *Frequent pattern mining: current status and future directions*, Data Mining and Knowledge Discovery **15** (2007), 55–86, 10.1007/s10618-006-0059-1.

[HW06] Pat Hayes and Chris Welty, *Defining n-ary relations on the semantic web*, April 2006.

[Kaa97] Heiki-Jaan Kaalep, *An estonian morphological analyzer and the impact of a corpus on its development*, Computers and the Humanities, Kluwer Academic Publishers, 1997, pp. 115 – 133.

[Kar90] Fred Karlsson, *Constraint grammar as a framework for parsing running text*, COLING '90 Proceedings of the 13th conference on Computational linguistics (Hans Karlgren, ed.), vol. 3, KVAL Research Institute for Information Science, Stockholm, Sweden, Association for Computational Linguistics Stroudsburg, PA, USA, 1990, pp. 168–173.

[KRSW09] Gjergji Kasneci, Maya Ramanath, Fabian Suchanek, and Gerhard Weikum, *The yago-naga approach to knowledge discovery*, SIGMOD Rec. **37** (2009), no. 4, 41–47.

[LW02] A. Liaw and M. Wiener, *Classification and regression by randomforest.*, R News: The Newsletter of the R Project **2** (2002), no. 3, 18–22.

[MPM$^+$03] Kaili Müürisep, Tiina Puolakainen, Kadri Muischnek, Mare Koit, Tiit Roosmaa, and Heli Uibo, *A new language for constraint grammar: Estonian*, International Conference Recent Advances in Natural Language Processing, Sep. 2003, pp. 304–310.

[NTW11] Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum, *Scalable knowledge harvesting with high precision and high recall*, Proceedings of the fourth ACM international conference on Web search and data mining (New York, NY, USA), WSDM ’11, ACM, 2011, pp. 227–236.

[PL11] Timo Petmanson and Sven Laur, *Mallipõhine faktituletus tekstikorpustest. esimene vaheraport.*, Tech. report, Eesti Keeletehnoloogia, 2011.

[PLB+06] Marius Paşca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain, *Names and similarities on the web: fact extraction in the fast lane*, ACL-44 Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics (Stroudsburg, PA, USA), Association for Computational Linguistics, 2006, pp. 809–816.

[SIW06] Fabian M. Suchanek, Georgiana Ifrim, and Gerhard Weikum, *Combining linguistic and statistical analysis to extract relations from web documents*, Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM New York, NY, USA, 2006, pp. 712 – 717.

[Sod99] Stephen Soderland, *Learning information extraction rules for semi-structured and free text*, Machine Learning **34** (1999), 233–272, 10.1023/A:1007562322031.

[SSW09] Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum, *Sofie: A self-organizing framework for information extraction*, 18th International World Wide Web Conference, April 2009, pp. 631–631.

[ST95] Daniel Dominic Sleator and David Temperley, *Parsing english with a link grammar*, CoRR **abs/cmp-lg/9508004** (1995).

[TCM99] Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney, *Active learning for natural language parsing and information extraction*, Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99) (Bled, Slovenia), June 1999, pp. 406–414.

[Tka10] Aleksandr Tkachenko, *Named entity recognition for estonian language*, Master’s thesis, University of Tartu, Estonia, 2010.

[Vö05] Johanna Völker, *Towards large-scale, open-domain and ontology-based named entity classification*, In Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP’05, INCOMA Ltd, 2005, pp. 166–172.

Resümee (eesti keeles)

Käesoleva töö eesmärgiks on mustripõhise faktituletuse prototüüprakenduse loomine eestikeelsete tekstide jaoks. Töö teoreetilises osas kirjeldame mustreid kui üldistatud sõnajärjendeid, mis suudavad tekstidest üles leida lõike ning lauseosi potensiaalselt olulise informatsiooniga. Praktiline lähenemine eeldab treeningkorpuse olemasolu, kus käsitsi on märgendatud mingi konkreetse relatsiooni jaoks need lauseosad, mis on informatiivselt olulised. Taolisel korpusel on võimalik teha mustrikaevet ning teoreetilises osas kohandame andmekaeves tuntud Apriori algoritmi nõnda, et seda on võimalik kasutada meie ülesande lahendamiseks vajalike mustrite leidmiseks. Samuti räägime lihtsast heuristilisest metoodikast, mis võimaldab mustrikaeve protsessi kiirendada.

Töös kirjeldame samuti masinõppe meetodeid, mida kasutame üksikute mustrite väljundi koondamiseks ning nende põhjal täpsemate ennustuste tegemiseks. Katsetame mustrikaevet koos masinõppe meetoditega mitmel reaalsel tekstikorpusel ning analüüsime tulemusi.

Lisaks kirjeldame aktiivõppe metoodikat, mis on abiks suurte tekstikorpuste märgendamiseks. Selgitame võimalusi, mis aitavad fookusest välja jätta hulga lauseid, kus otsitavat relatsiooni suure tõenäosusega ei esine või sarnaselt otsida üles näited, mis tõenäoliselt kirjeldavad meie relatsiooni. Analüüsime praktilise testi tulemusi, kus rakendame aktiivõpet mitmel näidiskorpusel.

Väljatöötud metoodika testimiseks implementeerisime prototüüplahendused, mis on kirjeldatud põhjalikumalt Lisas A ja millega viisime läbi praktilises osas tehtud testid.

Appendix A

The prototype implementation

We implemented prototype solutions for carrying out the case studies in the experiment. In fact, we ended up implementing two versions of the core component, having subtle differences. We call them version A and version B of the prototype. The main differences are described in the table below:

Version A	Version B
Uses complex patterns discussed in work [PL11].	Uses simple patterns.
Extraction areas are limited to single tokens.	Extraction areas can be one or more consequent tokens.
Fixed for binary patterns. Unary patterns can be emulated.	Implemented for n-ary patterns.
Statistical estimations calculate the amount of all possible negative matches in a corpus.	Statistical estimations assume one true negative match per sentence.

The reason for implementing two different systems is that we wanted to test different implementation strategies and learn the benefits and problems of both. Main problem of both prototypes was the slow execution. In version A, the main reason was usage of complex patterns. In version B, the usage of SQLite database as a querying mechanism for the covers (version A stored the deserialized corpus in computer memory).

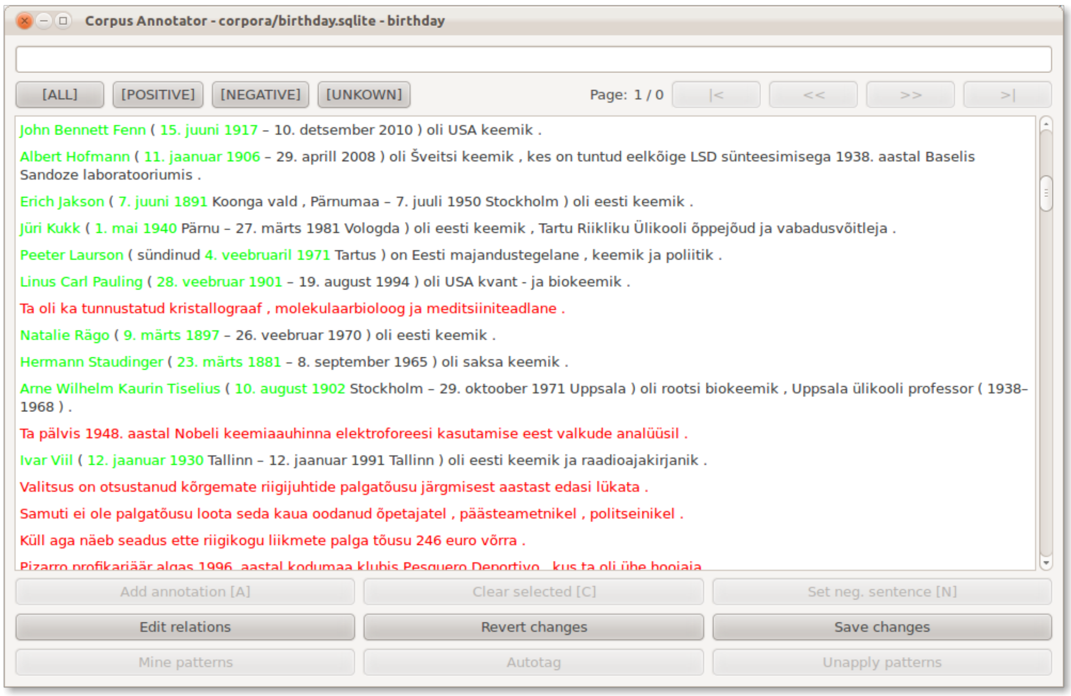
The preprocessing stack was also quite slow. The main reason is the usage of VISLCG3 component for syntactic analysis in the process. ESTMORF program itself was sufficiently fast. Also, additional steps were required to parse partly erroneous output from the preprecessing scripts and handling a bug in syntactic analysis grammar, that caused the VISLCG3 component to crash the system, although T.Puolokainen (the grammar author) was able to fix that particular error later using a sample of our failed text. Additionally, extra routines were required to merge NER annotations with the syntactic output, if NER was used. What we did not do in the preprocessing stack, was fixing some bugs in the preprocessing scripts of other authors. In several cases, the scripts worsened the output of the ESTMORF program by adding wrong attributes to the labeling. The syntactic labeling itself seemed reasonable, therefore we decided to keep the attribute. In case of ambiguity, the preprocessing scripts left all available options to be used. In these cases, we just chose the first option as our methods were not designed to work with ambiguous labeling.

Features.

- GUI tool for annotating positive and negative matches in the corpora, see Figure A.1 for a screenshot.
- Command line scripts for free-text preprocessing, model training and annotation, see Figure A.2 for a screenshot.
- Web front end for model exploration and online text annotation using either JavaScript GUI for web browser access or HTTP requests for XML output, see Figure A.3 for a screenshot.
- Preprocessing can optionally use ETMORF online web service, in case the user does not have the Filosoft license.
- Developed algorithms and methods can be used as an 3rd party library in other Python programs.

Used technologies. The prototype was mostly implemented in Python3 language, with support of various libraries. In particular, we used RPy2 to interface R language libraries for machine learning; we used PyQT4 for annotator GUI and CherryPy for setting up the web service. The preprocessing stack requires additionally Java platform, Perl, awk and bash to operate. Our code was written in cross-platform usage in mind, but we have only tested the implementation on three different Linux machines.

Packaging. The prototypes and all used corpora are included on a compact disk on the back cover of the work.



John Bennett Fenn (15. juuni 1917 – 10. detsember 2010) oli USA keemik .

Albert Hofmann (11. jaanuar 1906 – 29. aprill 2008) oli Šveitsi keemik , kes on tuntud eelkõige LSD sünteesimisega 1938. aastal Baselis Sandoze laboratooriumis .

Erich Jakson (7. juuni 1891 Koonga vald , Pärnumaa – 7. juuli 1950 Stockholm) oli eesti keemik .

Jüri Kukk (1. mai 1940 Pärnu – 27. märts 1981 Vologda) oli eesti keemik , Tartu Riikliku Ülikooli õppejõud ja vabadusvõitleja .

Peeter Laurson (sündinud 4. veebruaril 1971 Tartus) on Eesti majandustegelane , keemik ja poliitik .

Linus Carl Pauling (28. veebruar 1901 – 19. august 1994) oli USA kvant - ja biokeemik .

Ta oli ka tunnustatud kristallograaf , molekulaarbioloog ja meditsiiniteadlane .

Natalie Rägo (9. märts 1897 – 26. veebruar 1970) oli eesti keemik .

Hermann Staudinger (23. märts 1881 – 8. september 1965) oli saksa keemik .

Arne Wilhelm Kaurin Tiselius (10. august 1902 Stockholm – 29. oktoober 1971 Uppsala) oli rootsi biokeemik , Uppsala ülikooli professor (1938–1968) .

Ta pälvis 1948. aastal Nobeli keemiaauhinna elektroforeesi kasutamise eest valkude analüüsil .

Ivar Viil (12. jaanuar 1930 Tallinn – 12. jaanuar 1991 Tallinn) oli eesti keemik ja raadioajakirjanik .

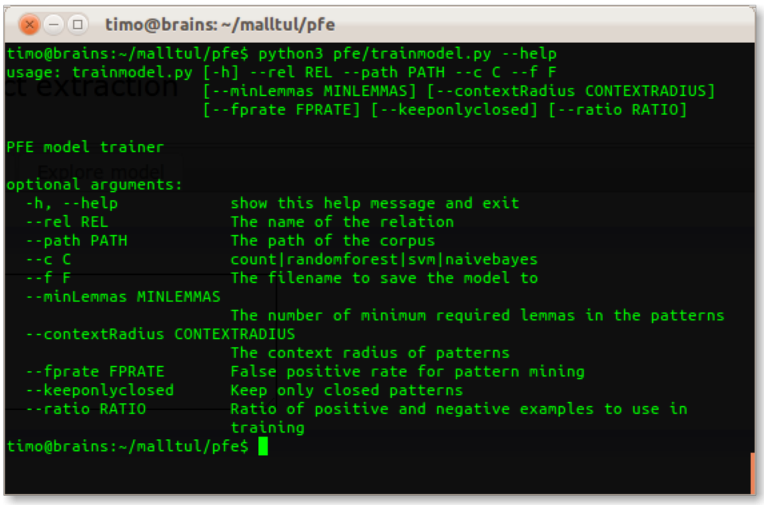
Valitsus on otsustanud kõrgemate riigijuhtide palgatõusu järgmisest aastast edasi lükata .

Samuti ei ole palgatõusu loota seda kaua oodanud õpetajatel , päästeametnikel , politseinikel .

Küll aga näeb seadus ette riigikogu liikmete palga tõusu 246 euro võrra .

Pizarro profikarjäär algas 1996. aastal kodumaa klubis Pesquero Deportivo , kus ta oli ühe hooaja

Figure A.1: Screenshot of the prototype annotator GUI application. The program allows to define relations in a corpus and mark positive and negative examples.

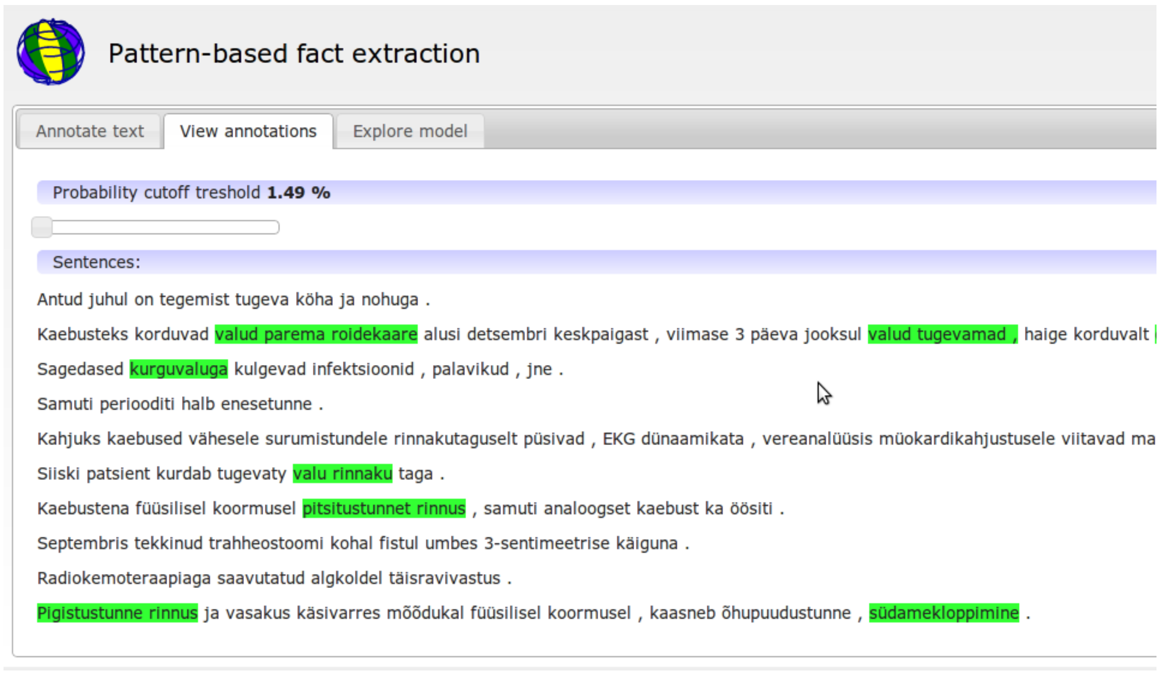


```
timo@brains:~/malltul/pfe$ python3 pfe/trainmodel.py --help
usage: trainmodel.py [-h] --rel REL --path PATH --c C --f F
                     [--minLemmas MINLEMMAS] [--contextRadius CONTEXTRADIUS]
                     [--fprate FPRATE] [--keeponlyclosed] [--ratio RATIO]

PFE model trainer

optional arguments:
  -h, --help            show this help message and exit
  --rel REL             The name of the relation
  --path PATH           The path of the corpus
  --c C                 count|randomforest|svm|naivebayes
  --f F                 The filename to save the model to
  --minLemmas MINLEMMAS
                        The number of minimum required lemmas in the patterns
  --contextRadius CONTEXTRADIUS
                        The context radius of patterns
  --fprate FPRATE       False positive rate for pattern mining
  --keeponlyclosed      Keep only closed patterns
  --ratio RATIO         Ratio of positive and negative examples to use in
                        training
timo@brains:~/malltul/pfe$ 
```

Figure A.2: Screenshot of model training program command line interface. The prototype contains command line tools for plaintext linguistic analysis, model training and annotation.



Probability cutoff treshold **1.49 %**

Sentences:

Antud juhul on tegemist tugeva köha ja nohuga .

Kaebusteks korduvad valud parema roidekaare alusi detsembri keskpaigast , viimase 3 päeva jooksul valud tugevamad , haige korduvalt

Sagedased kurguvaluga kulgevad infektsioonid , palavikud , jne .

Samuti perioodití halb enesetunne .

Kahjuks kaebused vähesele surumistundele rinnakutaguselt püsivad , EKG dünaamikata , vereanalüüsis müokardikahjustusele viitavad ma

Siiski patsient kurdab tugevaty valu rinnaku taga .

Kaebustena füüsilisel koormusel pitsitustunnet rinnus , samuti analoogset kaebust ka öösiti .

Septembris tekkinud trahheostoomi kohal fistul umbes 3-sentimeetrise käiguna .

Radiokemoteraapiaga saavutatud algkoldel täisravivastus .

Pigistustunne rinnus ja vasakus käsivarres mõõdukal füüsilisel koormusel , kaasneb õhupuudustunne , südamekloppimine .

Figure A.3: Screenshot of prototype's web interface. The interface can be used to annotate text, retrieve them in XML or view the HTML format and explore the patterns of the model.