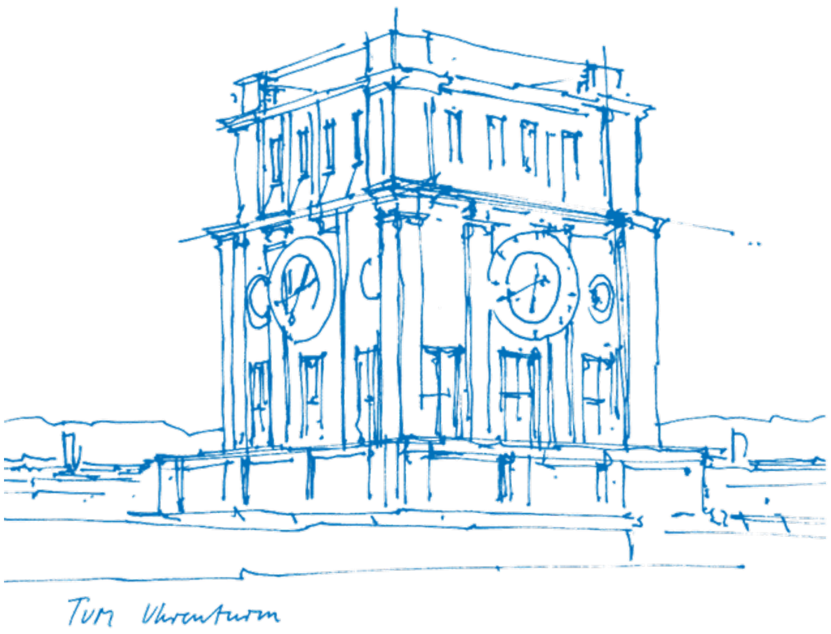
Chair of Information Systems and Business Process Management (i17)
Department of Computer Science
TUM School of Computation, Information and Technology
Technical University of Munich

Bachelor's Thesis in Information Systems

Jacob Fehn

Automated Change Identification and Classification for Legal Documents and Their Amendments



Chair of Information Systems and Business Process Management (i17)
Department of Computer Science
TUM School of Computation, Information and Technology
Technical University of Munich

Bachelor's Thesis in Information Systems

Jacob Fehn

Automated Change Identification and Classification for Legal Documents and Their Amendments

Automatische Erkennung und Klassifizierung von Änderungen in Rechtsakten

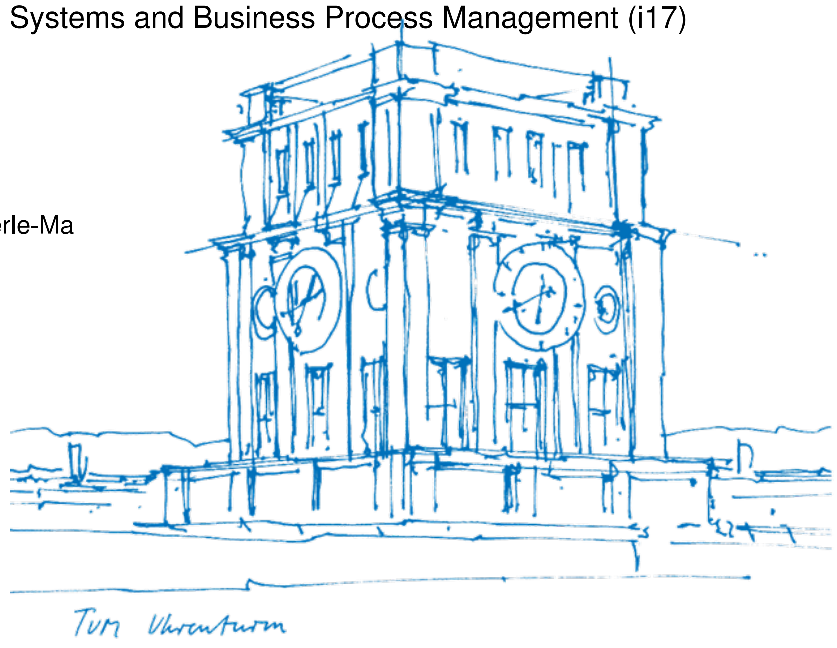
Thesis for the Attainment of the Degree
Bachelor of Science

at the TUM School of Computation, Information and Technology,
Department of Computer Science,
Chair of Information Systems and Business Process Management (i17)

Examiner
Prof. Dr. Stefanie Rinderle-Ma

Supervised by
Catherine Sai

Submitted on
15.11.2023



Declaration of Academic Integrity

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.

This thesis was not previously presented to another examination board and has not been published.

München, 15.11.2023 Jacob Fehn

Abstract

Every business process underlies the obligation to comply with the law. Fulfilling this obligation is complicated by the fact, that the legal documents are changing. These changes are not predictable. Not how often something changes, not in what period, not how much is changed. The aim of this bachelor thesis is, to classify possible changes in legal documents and the pattern those updates are following and to create a web service to extract the contained information using the knowledge gained by the classification. A set of two legal documents can be referenced via a link as input into the web service and the information in the detected changes will be displayed and formatted to be processed by business management software. This would reduce the cost of manpower needed to comply with the law and reduce the risk of extra expenditures by failing to adapt to a legal change. The algorithm in the backend of the web service uses natural language processing based on patterns and knowledge gained by prior classification to extract information from the text and only the changed text, not the whole document. The web service and its algorithm work with an accuracy of 98.65 percent in detecting modifications. To calculate this accuracy, a data set of 271 documents from EURLex and 594 individual tests were collected. The calculated values of Precision and Recall for identifying changes and modifications are all above 0.990. In total, the archived artifacts of this paper are an available set of data (*instantiation*) of 271 documents, a classification of 6 types of modifications as well as a thorough analysis of patterns commonly found in EU legislation (*model*) and a web service and its algorithm for functionality (*method*).

Keywords: *changes in legal documents, natural language processing, legal information extraction*

Contents

List of Tables

List of Figures

Introduction

For major companies, spanning a wide range of business fields, the overview of legal regulations and constraints can be difficult to keep up with. Every time, a legal document gets modified, experts have to compare, check and evaluate, whether the changes in law apply to the company's field of business, how to enforce them and to ensure, that their business processes are within legal boundaries. Manpower is a costly method to check for this compliance with legal constraints (for example data protection measures for stored data) and leaves room for human errors, which could lead to fees or other expenses for damage control. This checking must be conducted for every business process inside a company, which is called business process compliance (BPC[1]). An exemplary business process is registering on a website, after which the user data is stored for identification. In this case, the company running the website has regulatory rules to fulfill. Legally, the regulations from GDPR[2] concerning the security and safety of personal data must be put into practice. A violation of the GDPR (depending on the severity of the infringement) could result in fines up to €20 million, or 4 percent of the firm's worldwide annual revenue[3]. Additionally, the company can expand the regulations for the process with organizational rules (like a folder structure for the stored data) or rules to support Corporate Social Responsibility (CSR), which is the self-regulation of companies to support societal goals. A regulatory document for such a case could be ISO/IEC 27001[4], which is a common standard for information security. Complying with ISO27001 would assure legal safety in a business process storing user data, thus BPC is achieved. Research and literature on BPC and its management are evaluated in [1]. The paper concludes, that even though a great deal of work addresses the problem, there are still unsolved challenges, like formalizing and extracting norms. This thesis tries to solve the frequently changing legal environment by extracting only the changed information based on the format in which changes occur.

[1]abbreviation by [1]
[2]document 32016R0679 on `https://eur-lex.europa.eu/homepage.html` (Last access date: 07.11.23)
[3]`https://gdpr.eu/fines/` (Last access date: 07.11.23)
[4]`https://www.iso.org/standard/27001` (Last access date: 07.11.23)

Motivation

Given how expensive[5] a violation of regulations for stockholders could be, non-compliance should be avoided. This means, that with every legal change, a thorough check is needed. The company's regulatory documents might have to be adjusted to abide by the underlying legal document.

In numbers: alone in 2022 the European Union adopted amending acts of 27 regulations, 9 directives and 1 decision[6]. This is overall just barely under the average amount from the last 10 years, which is 38.3 amending acts per year. In Table 5 in the Appendix the adopted changes in 2022 are analyzed with the tool wordcount.com[7]. The on EURLex[8] published amending legal acts display the modification and sometimes the reasoning for the change. The average of nearly 5000 words per amending act shows, that the required effort is not negligible.

In the context of compliance regulatory documents are used as the basis for a business process. Regulatory documents are the formal guidelines that a business process follows. In those the principles of legal acts (like the ones on EURLex[8]) are implemented to ensure the process is legal. Regulatory documents also include non-legal rules. Those optimize organizational matters or social acceptance (CSR). Thus regulatory documents are not necessarily enforceable in court in contrast to legal documents. Those are formulated and entered into force by parliament containing acts of laws. This paper focuses on legal documents from EURLex[8], mostly EU regulations, -directives and -decisions to be precise. Those are legally binding acts, relevant for companies and enforceable in court. However, the output of the thesis can be used to adapt regulatory documents to changes in legal documents.

EURLex[8] displays legal documents either as a legal act, as a consolidated text (a merged document consisting of the original act as well as the amended parts over the years) or as an amending act, the form, in which a change is decided upon in parliament. EURLex provides a manual regarding the methodology of consolidation[9]. The question for a company's legal analyst is, which amendments

[5]another source: `https://saviynt.com/the-true-cost-of-non-compliance/` (Last access date: 07.11.23)

[6]`https://eur-lex.europa.eu/statistics/2022/legislative-acts-statistics.html` (Last access date: 07.11.23)

[7]`https://wordcount.com` (Last access date: 07.11.23)

[8]`https://eur-lex.europa.eu/homepage.html` (Last access date: 07.11.23)

[9]`https://eur-lex.europa.eu/content/intro/collection/Methodology-on-Consolidation.pdf` (Last access date: 07.11.23)

came into effect since the last check-up and what contents are relevant for the company and the business process in question.

This bachelor thesis tackles the question, of how to automatically find out which amendments came into effect since the last check-up. For example, if the last time a legal analyst for a chemical company checked REACH (EURLex 32006R1907 - Registration, Evaluation, Authorisation and Restriction of Chemicals) was in March 2023, then there are new 3 amending acts, shown in Figure 1, which could drastically impact the company's work if a chemical gets restricted or needs further authorization. The analyst would have to compare the last document used (17.12.22) to the newest consolidated version (06.08.23), find the 3 new changes (M72 with 1 modification, M73 with 5 modifications and M74 with 2 modifications) and check all 8 modifications with the old text from 2022. Important is the distinction between change and modification. A change contains all the modifications from one amending act, which is referenced at the beginning of a document. Modifications are instances of a change and are labeled accordingly, to make the connection between modification and change clear. This thesis wants to reduce the work of researching how many amendments were released since the latest legal check and simultaneously extract the updated information. For that, the different kinds of changes and their modifications in legal documents will be classified based on patterns. These classes will help create a web service that outputs the difference between two legal documents using an algorithm based on the knowledge gained by the classification. This will rely on natural language processing (NLP) to extract machine-readable information from the natural language used in legal documents. The output will show what contents have been added, what contents have been deleted and what contents have been altered between the two versions of legal documents. Figure 2 shows a rough walk-through of which steps the web service would take.

Known types of legal updates: every modification of text can be one of three types: ADD, DELETE and UPDATE. The paper [2] categorizes legal changes in *Replacement (From, To), Replacement-ref, Addition, Repeal and Abolition*. More detailed classification is found in [3] with *Substitution, Integration, Repeal, Variations, Derogation, Range, Modification of terms, Validity, Execution and others*. Others go back to the basic textual methods like in [4], where modifications are *replacement, insertion* and *repeal*, so to say UPDATE, ADD and DELETE. More detailed explanations are found in Section Changes in Legal Documents. Finding a suitable typification for legal changes will be needed as the basis for the web services algorithm and worked out in Chapter Solution Design.

Figure 1
Use-case diagram of an analyst performing a compliance check via this thesis' web service.

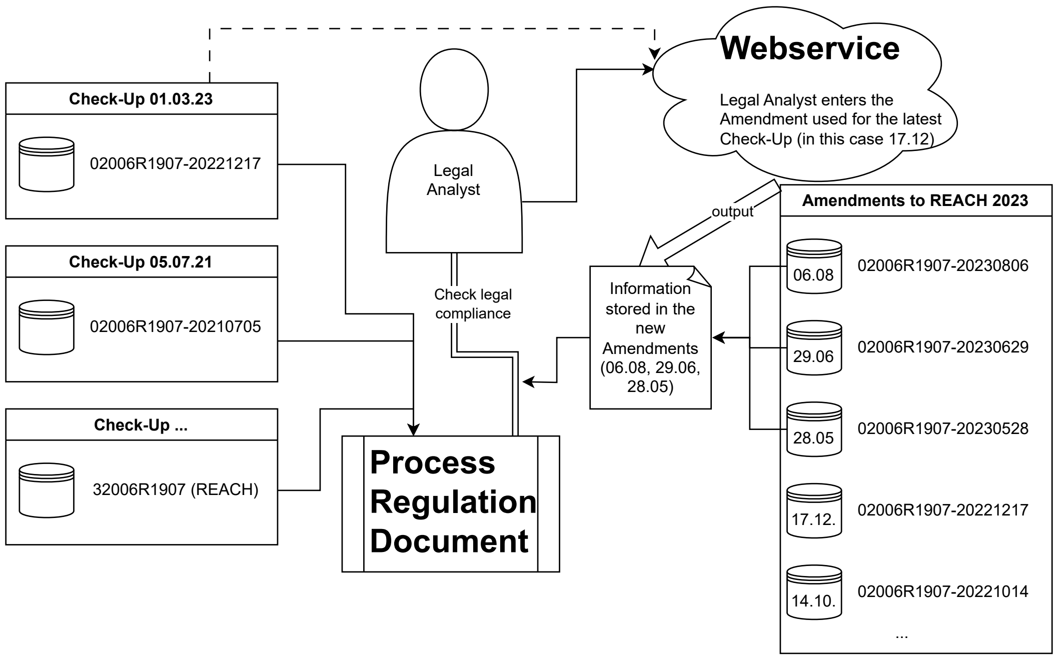
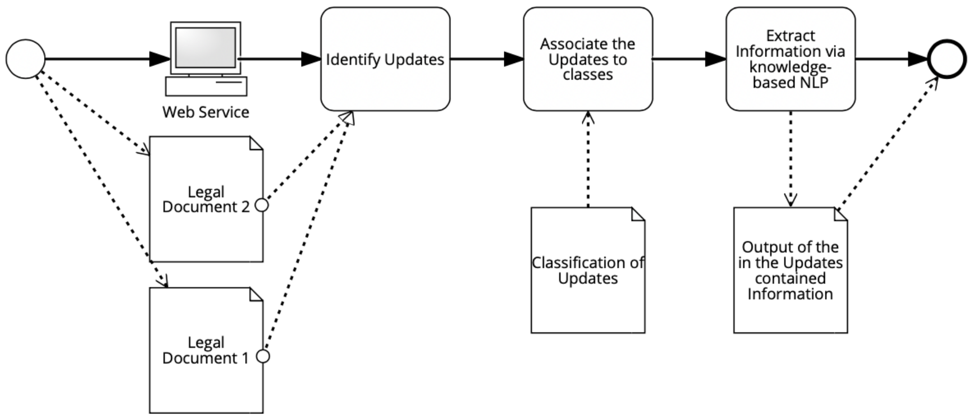

Webservice
Legal Analyst enters the Amendment used for the latest Check-Up (in this case 17.12)
Check-Up 01.03.23
02006R1907-20221217
Legal Analyst
Check-Up 05.07.21
02006R1907-20210705
Check-Up ...
32006R1907 (REACH)
Check legal compliance
output
Information stored in the new Amendments (06.08, 29.06, 28.05)
Process Regulation Document
Amendments to REACH 2023
06.08 02006R1907-20230806
29.06 02006R1907-20230629
28.05 02006R1907-20230528
17.12. 02006R1907-20221217
14.10. 02006R1907-20221014
...

Figure 2
This business process diagram shows the process behind the web service.


Web Service
Identify Updates
Associate the Updates to classes
Extract Information via knowledge-based NLP
Legal Document 2
Legal Document 1
Classification of Updates
Output of the in the Updates contained Information

Research Questions

- Which **patterns** are found in changes of legal documents on EURLex[8]?
- How can these patterns be **classified** and **used** to support information extraction?
- What NLP **techniques** or **approaches** are most suitable to extract data from changed text?
- How can changes be **displayed** to aid hybrid systems for legal business compliance?

Research Methodology

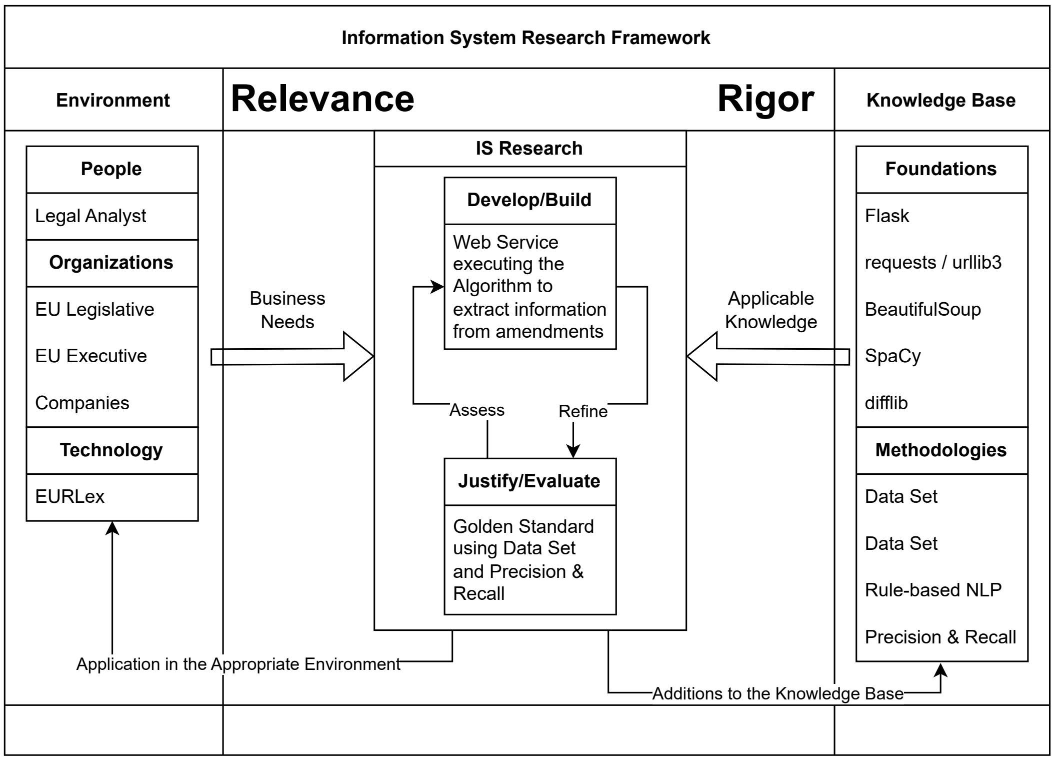
The artifacts that will be produced are an available set of data (*instantiation*), containing binding legal acts with consolidated versions from EURLex[8], classification of changes for legal acts (*model*) and a web service providing the functionality of information extraction from legal changes (*method*). Following the seven guidelines from [5] on page 83 as well as the framework from page 80, the artifacts will be designed, as shown in Figure 3. Roughly: (1) all artifacts will be produced in a correct form, (2) the relevance of the problem is explained in Section Motivation, (3) the evaluation method is explained in Section Evaluation, (4) the contribution is explained in Section Contribution, (5) the method of constructing and evaluating will be strict and well documented, which should be rigor, (6) the start of the search process is told in Chapter Related Work and (7) the communication will be achieved by publishing the thesis paper as well as the corresponding project per git-repository[10].

Structure

The structure of this thesis starts with a Motivation, which is needed as an incentive to find a solution to the problem. This problem poses certain questions - Research Questions - which need to be researched and answered properly. In order to do this properly, a Research Methodology must be established, which defines the working guidelines for the Solution Design. Important for the solution is the Related Work to distinguish between state of the art and own Contribution. Also the Related Work gives an overview of options that can help, solve the problem. This Solution Design the Implementation has to follow. The success of Solution Design and Implementation is explained in the Evaluation and the Discussion afterward goes further into Challenges. After all that, a Conclusion can be drawn.

[10]`https://github.com/affentypi/Webservice_Thesis.git`

Figure 3
The Information System Research Framework by Hevner([5] p.80) filled with the thesis' artifacts.



Related Work

For research, DBLP[11] and Google Scholar[12] were used as engines for keyword searches. In Table 1 the keywords and their corresponding matches are listed along with the used search engine. Especially the conferences JURIX[13] (2013 - 2022) and ICAIL[14] (2011 - 2021) were scanned for interesting papers.

Criteria for **inclusion** were:

- pattern- or knowledge-based NLP approaches for information extraction in a legal context
- working with EURLex[8]
- patterns in legal documents and classifications

[11]`https://dblp.org` (Last access date: 07.11.23)
[12]`https://scholar.google.com` (Last access date: 07.11.23)
[13]`https://dblp.org/db/conf/jurix/index.html` (Last access date: 07.11.23)
[14]`https://dblp.org/db/conf/icail/index.html` (Last access date: 07.11.23)

Table 1
A relation of keywords searched and matches found as well as a rating of the found results. In the column ratings, 'o' stands for overflow, meaning not every result could be reviewed in detail. Furthermore,'n' stands for mostly not related results and digits indicate the number of useful papers saved to this bibliography.

Keyword	Matches	Engine	Rating
"*legal modification*"	5	DBLP[11]	(1)
"*legal modifications pattern regular expressions nlp*"	20600	GoogleScholar[12]	o(1)
"*nlp legal modification patterns extraction*"	24100	GoogleScholar[12]	o(3)
"*EUR-lex*"	139	DBLP[11]	n(1)
"*eur-lex information extraction nlp pattern based*"	1440	GoogleScholar[12]	o(2)
"*regulatory documents updates*"	1	DBLP[11]	(1)
"*classifying Legal*"	10	DBLP[11]	(1)
"*regulatory documents*"	30	DBLP[11]	(1)
"*regulatory documents information extraction nlp*"	22800	GoogleScholar[12]	o(2)

Criteria for **exclusion** were:

- case-law
- text analysis of arguments and reasonings
- translations or summaries of legal documents

The following literature was found through these searches. Literature was selected first based on the title; if the headline matches the topic. Secondly, they were filtered by their abstracts and citations and lastly based on the content after reading through the still-relevant papers.

Changes in Legal Documents

The paper [2] approaches automated consolidation of legal documents by classifying textual modifications. Here, one possible classification of legal amendments on documents in the European Union is presented. While in this paper, the authors use their findings to automate the process of consolidating legal acts with later amendments, this thesis will use the patterns for a systematical extraction of information of only the amending acts. The paper focuses on the Italian translations of legal acts but it also uses EURLex[8]. The following patterns were found:

- *replacement* of small parts, like expressions, words or dates, or big parts, like articles, paragraphs or indents. It usually consists of *from*, what was there before, and *to*, what is there afterward (UPDATE).
- *replacement-ref* as a type of replacement containing attachments (UPDATE).
- *addition* adds a small or big part to a sentence or document (ADD).

- *repeal* describes the removal or reversal of a legal act; all concerned provisions are revoked (DELETE).
- *abolition* indicates the removal of a part of a legal act while keeping the provisions (maybe updated, amended or related) intact (DELETE/UPDATE).

Additionally, [2] works on feature extraction using an Italian model of SpaCy[15]. SpaCy is a NLP library, which will also be useful for this paper. The used technique of extraction works on word level, and uses, inter alia, n-grams. The idea of n-grams is to separate texts into small linguistic items in a contiguous sequence and count the repetitions. The presented machine-learning approach will not be relevant for this thesis because as in [6] recommended, a pattern-based classifier will be used. Machine learning can classify pretty accurately, but is more complex and has further problems, while pattern-based (a method relying on knowledge gained in advance) gives clear distinctions between classes and optimizes the process using prior knowledge. For automation usage of machine-learning tools like Support Vector Machine (SVM), a machine-learning algorithm introduced in [7], is not precluded for future work, but in this thesis, the extraction will work with manually entered documents or a set of data for testing and will process them by an iterative algorithm.

The paper [8] works on an automated prediction, of whether updates of regulatory documents are significant or not. While the technique is less useful because all changes, significant or less significant, are relevant for information extraction, the presented way of working with unigrams, size 1 n-grams was of interest. In the paper, the n-gram items are 'bag of words' sets, which are viewed independently, without taking the following 'bags of words'(BOW) into account. This way, a once-conducted extraction from a BOW can be applied for every repetition of this BOW. This thesis did not end up using n-grams but researching it led to another useful paper: [9] create a set of data by crawling the website starting from the domains and then filtering the documents using among other things n-grams. The acquisition of data from EURLex[8] is interesting for later work for the Test Data.

[3] presents the TULSI system, which extracts legal 'modificatory provisions' in the Italian language. It first presents the structure of 'modificatory provisions' (amending acts, which change/modify legal documents): ActiveNorm, PassiveNorm, Action, Times, Content, Purview, Space, Conditions.

[15]`https://spacy.io` (Last access date: 07.11.23)

Furthermore, for the syntactical analysis, a rule-based parser named TUP[16] is used. Using chunking, coordination and verbal subcategorization, TUP connects dominant words (the 'head') with the 'dependent' like in "Joe sleeps" the head "sleep" with the dependent "Joe". Except for this method of analysis, there are the classes of modifications presented in [3]: (sorted by frequency of occurrence)

- *Substitution* replaces text with a new text (UPDATE).
- *Integration* adds text (ADD).
- *Repeal* deletes text (DELETE).
- *Variations* replaces or adds small parts of text (ADD/UPDATE).
- *Derogation* deletes small parts of text (DELETE).
- *Range* is the change of parameters in a text (UPDATE).
- *Modification of terms* is the change of wording (UPDATE).
- *Others* like *Validity* or *Execution*.

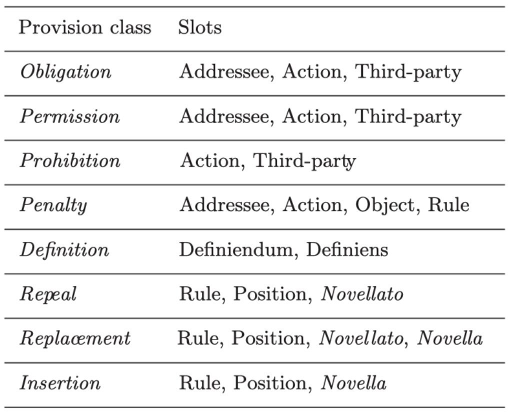
Functional Types of Legal Statements

Paper [10] classifies legal sentences in German lawmaking (which means it is not about case law) based on semantics. It uses machine learning, which won't be used in this thesis. Still, the paper can offer insights into feature extraction (this time solely via TFIDF explained in [11]) and the usage of SpaCy[15]. Also, the presented sentence types are called "beneficial for information retrieval", which describes the information extraction this thesis aims at. In [12] those sentence types are classified more precisely. The paper also classifies legal documents in German lawmaking using a machine-learning approach. The functional classes of legal statements are roughly divided into 4 types of statements: *normative, auxiliary, legal-technical, legal-mechanism*, with more detailed smaller subtypes. Both, [10] and [12], present suitable and useable types of sentences which can be used as prior knowledge in NLP.

[4] presents similar types but for provisions, not sentences. Provisions are divided into 3 main categories, *obligations, definitions* and *modifications*. The last one includes the classes *replacement, insertion* and *repeal* like in Section Changes in Legal Documents and is comparable to the modifications worked on in this thesis. *Obligations* are further divided in *obligation, permission, prohibition* and *penalty*. These are overall 8 types (4 *Obligations* + 3 *Modifications* + 1 *Definiton*), which are

[16]https://www.slideserve.com/shae/the-rule-based-parser-of-the-nlp-group-of-the-university-of-torino (Last access date: 07.11.23)

Figure 4
Slots corresponding to the provision types for a frame-based description by [4]. The figure is from the related public full text on `https://www.researchgate.net/publication/221539373_Automatic_semantics_extraction_in_law_documents` *(Last access date: 08.11.23) on page 263 (Table 1).*



Provision class	Slots
Obligation	Addressee, Action, Third-party
Permission	Addressee, Action, Third-party
Prohibition	Action, Third-party
Penalty	Addressee, Action, Object, Rule
Definition	Definiendum, Definiens
Repeal	Rule, Position, *Novellato*
Replacement	Rule, Position, *Novellato*, *Novella*
Insertion	Rule, Position, *Novella*

represented frame-based using slots, see Fig. 4. This taxonomy (also referenced in [13]) comes from the SALEM framework, which both papers use. This framework is used to extract information using NLP. The approach is using a frame-based legal ontology (said types of provisions) to analyze the semantics of legal documents. The architecture of SALEM itself according to [13] uses chunks of text, just like BOW, to represent the text. The "semantic mark-up component" of the architecture then uses regular expressions on chunks of text to find the patterns based on the provision types from Figure 4. The mentioned 'frame-base' is based on knowledge about the ontology of legal documents and uses the pattern provisions they follow. [4] gives examples for that. The SALEM architecture must first assign a paragraph to one of the provision types and secondly tag legal entities (like actors, actions or properties) from parts of the paragraph based on domain-specific semantic roles. An identified replacement frame for example would consist of 5 slots: FRAME (the type), RULE (the text being modified), POSITION (the position, where new text is inserted), NOVELLATO (the old text) and NOVELLA (the new text). These frames can help represent and understand the information to be extracted. The framework of the SALEM architecture further works with preprocessing and parsing of texts, which the following Section Information Extraction by Natural Language Processing is about.

Information Extraction by Natural Language Processing

In [14] a framework to extract a formal description from regulatory documents is presented. The information from the regulations is extracted and checked for consistency as well as whether implementations conform to them. The regulations refer to blood banks in America, whereas this present paper will concentrate on legal documents from the European Union, but the idea behind the framework still applies. Using Abstract Syntax Trees (ASTs) to distinguish between obligation and permission (or possibly more types like those listed in Functional Types of Legal Statements) could be used for future work as explained in Conclusion. An AST could represent a BOW and simplify working with it and extracting information because actual data will be stored in the end nodes. Additionally, [15] shows, that ASTs can be useful to represent the logical structure of statements. There, ASTs are used to display logical formulas underlying natural English legal documents. In that way, ASTs can be used to also understand the logic of a sentence. Another way of Information Extraction is shown in [16]. The proposed approach uses semantic, rule-based NLP to extract information automatically from regulatory documents. A rule-based approach uses manually coded rules for text processing, based on prior knowledge (*pattern-based*). The paper [16] affirms, that a pattern-based NLP performs better in text processing (in terms of precision and recall), something [6] also stated. In addition to information extraction rules, the paper uses rules to resolve possible conflicts. [16] proposes tuples to represent information, so-called "semantic information elements", a tuple consisting of 1) an ontology concept; 2) an ontology relation; 3) a deontic operator indicator: which type like those in Functional Types of Legal Statements. Those elements are distinguished as simple (rigid) or complex (flexible). Simple semantic information elements are rigid and consist of a single concept/relation/indicator, while complex ones are flexible and consist of several concepts and relations, which is needed for full-sentence analysis. The paper itself uses for its application a 9-tuple representation but still follows the idea of a tuple with these 3 main parts. This is another way of possible representation of information next to the previous 'frame-based' one. Further in [16], the method of Information Extraction is shown, with Preprocessing (consisting of tokenization, sentence splitting, morphological analysis and De-hyphenation) and Feature Generation (consisting of part-of-speech (POS) tagging, phrase structure analysis using phrase structure grammar (PSG), gazetteer compiling and ontology-based semantic analysis). Those concepts can be useful for this work and are part of the SpaCy[15] pipeline from Figure 8.

Since it is established, that this thesis will approach the extraction of data via patterns, the paper [17] can be taken into account. The paper combines syntax- and logic-based patterns, to extract information. The syntax patterns are found using the Standford Parser[17], a natural language parser, that works out the structure of a sentence, e.g. the subject or object of a verb. This is part of

[17]`https://nlp.stanford.edu/software/lex-parser.shtml` (Last access date: 07.11.23)

SpaCy[15], so the Standford Parser was dismissed as a library. A similar dependency pattern could be found in [18]. The rules in the proposed pattern concentrate on the relationship between incident and measure. Based on phrase patterns the relation between an incident and a measure can be found (e.g. "protection from X" indicates a direct measure and "prevention from X" indicates a preventive measure). Those phrases are found with the syntactical analysis of structural dependencies in a sentence, where the subject, verb, auxiliary verb, object etc. are identified and brought in relation. This has to be carried out for every relevant sentence in the text; for this thesis, every sentence in the modification would be relevant. Also, an algorithm for co-occurrence and pattern-based relation extraction from regulatory documents is presented, which helps to identify already extracted semantic concepts to avoid double execution. This was not done in this thesis but is relevant for future optimization. [18] also uses SpaCy[15] for preprocessing.

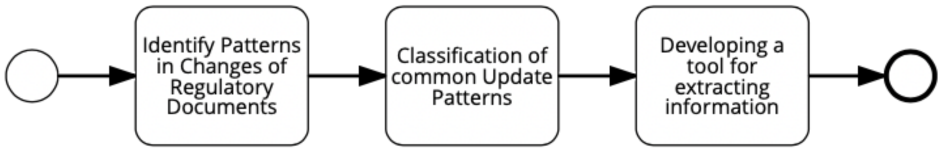
The paper [19] uses a mixed approach to extract information out of legal documents from EURLex[8]. The tools used are: the BOW representation, machine-learning algorithms like decision trees, the naive Bayes algorithm and SVM (those are less relevant for this thesis but mentionable covering the paper) and a syntactic parser (in this case PALAVRAS[18]). The used dataset is not suitable for this thesis, because the documents obtained are "international treaties" and not legal acts. More relevant is the named entity extraction (for which these tools are used). Names of people, places, organizations, products, dates, dimensions and currencies are located in the text and used to fill attributes of a template or to locate a sentence containing such entity. For this, morphological and syntactic information levels are analyzed. This thesis will also work on recognizing and extracting entities to display them in the web service.

Compliance

The paper [20] works on automating compliance based on semantic rules. Compliance is not per se the topic of this thesis, because it relies on regulatory documents and not on legal documents, but the principles they follow are nearly the same: something is forbidden, allowed or obligatory. This means the semantic pattern will be similar (if not the same) and complying with legal acts is of course essential. The parts concerning compliance checking will be less relevant but the underlying textual and ontological work is interesting. Therefore a semantic framework is used, which consists of 3 elements: understanding the semantics of the domain, of the regulation and the

[18]https://docslib.org/doc/12107632/the-palavras-parser-and-its-linguateca-applications-a-mutually-productive-relationship (Last access date: 07.11.23)

Figure 5
This (very simple) diagram shows the process plan of the thesis.


Identify Patterns in Changes of Regulatory Documents
Classification of common Update Patterns
Developing a tool for extracting information

related data file formats. The generic analysis for that presents the Abstract Regulation Ontology, for the elements then: *Core Domain Ontology*, *Regulation Ontology*, *Data Format Ontology* and for the relation of regulation and data format *Regulation Mapping Ontology*. For the Extraction of rules **Application** (which restricts the Scope), **Selection** (which increases the Scope), **Exception** (which allows the specification of exceptions to the rule being specified), and **Requirement** (which specifies the definitive requirements that must be met) from RASE out of [21] are used. Those principles have to apply to every rule, so they can be used to filter through all possible rules.

Solution Design

The development plan in Figure 5 shows the general process for this thesis in an oversimplified form. Aside from that, data has to be collected. This data consists of legal documents from EURLex which were amended once or more and have consolidated versions of those amendments. Those will work as a golden standard. Starting the development, a concept for an algorithm has to be created, on which basis the web service can be created. This concept will be found by comparing the amending acts and finding common grounds (Fig. 5: first box), in which the different amendments can be categorized (Fig. 5: second box). Based on these categories an algorithm can be put together (Fig. 5: last box), where those common grounds can be used to extract only the added, updated or removed information and not the whole document. To finish the web service, a frontend must be created, where a user can input two links to legal documents and will receive the differences as machine-readable output.

Classification

The first step is finding the data set. This process should result in first insights into the structure of the documents and their changes. But mainly a golden standard for a quantitative test is laid down.

The data needs to be collected by hand as explained in Section Test Data. EURLex[8] already explains consolidated versions[19] where the "kind" of changes are mentioned: B for the basic or initial act, M for modifiers (like amending acts), C for corrigendi (corrections for example of references or translations) and A for accession treaties (for new countries). Observations were, that "M-Changes" are by far the most common, followed by "C-Changes" with "A-Changes" being rare, which might not be very surprising. More important was the detail, that the change names (like M1 or C1) have an additional identifier before the name for every modification. There needs to be noted, that changes are a collection of text modifications, which were all set into effect by the same "changing act". Modifications are instances of the change itself. These identifiers in front of the change names are "►" and "▼" and only occur in the table of changes ("Amended by"/"Corrected by") if the changes are still found in the consolidated version because some modifications get overridden by later modifications and thus are not depicted anymore. The meaning of these arrows is not clarified by EURLex[8]. But the observed meaning is that "►" are modifications inside a "block of text" or only a single line. Except for in the table of changes this arrow is always followed by "◄", which indicates the end of this modification. Modifications with the "arrow" ▼" span over a "block of text" and their ending is indicated by another change or the basic act continuing, which both are again indicated by the "arrows".

This results in two first symbolical patterns that can be used to extract the information: the whole text can be categorized in ►B, ►M or ►C passages, which are indicated by a unique symbol not used otherwise in the legal act. This will be used in the implementation to find the modifications. A semantic meaning is barely extractable except for the size of the modification.

The basic legal act (like here for REACH[20]) shows under "Relationship between documents" all modifications as "modified by" and categorizes them under the comment as "*Addition*", "*Amendment*", "*Completion*", "*Deletion*", "*Repeal*" and "*Replacement*". There could be more but those are neither described nor documented by EURLex[8]. Those can be used to semantically categorize the changes but assigning the table entries to the modifications in the text can be difficult because the only reference to the position in the document is a textual description of the "subdivision concerned" and additionally the CELEX number of the change (which could be assigned to "change names").

[19] `https://eur-lex.europa.eu/collection/eu-law/consleg.html` (Last access date: 07.11.23)

[20] `https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32006R1907` (Last access date: 09.11.23)

This severely limits the usage of this table. Moreover, some files (like this 02012R0211-20171211[21]) do not have this table but just a link to an EURLex search, where all documents related to the given document are found.

On a syntax basis, the changes do not differ, because if something was added or replaced is only visible, if the old document is opened in a second tab and the text can be visually compared side by side, thus a more thorough approach is needed, as aimed at in this thesis. The only syntactic obvious change from the EURLex[8] is "*Deletion*" or "*Repeal*" because they are marked by "——————" behind the "change name". To compare "*Addition*" or "*Replacement*", the only way to solve by code is to find the sentences before and after the change in both documents and check what is in between. "*Amendment*" is an exception, where there is text in the middle, but it is nearly the same as before (which can be calculated by cosine similarity as described in [22]) or it is added text, looking like an "*Addition*". Because the exact definition is not provided by EURLex[8], those classifications are useful as names, but not as categories to sort the modifications by, because assumptions about the meaning would be needed as definitions.

In Chapter Changes in Legal Documents classifications used in related work are mentioned. The paper [2] categorizes legal updates in *Replacement (From, To), Replacement-ref, Addition, Repeal* and *Abolition*. More detailed classification is found in [3] with *Substitution, Integration, Repeal, Variations, Derogation, Range, Modificationof terms, Validity, Execution and others*. [4] defines modifications as *replacement, insertion* and *repeal*, which basically describes the main types of textual modifications: ADD, DELETE and UPDATE. Those are again similar to "*Addition*", "*Deletion*" and "*Replacement*" from EURLex[8].

Conclusively, using "*Addition*", "*Deletion*" and "*Replacement*" is a valid and easy classification to use for the implementation. The gainable knowledge of recognizable patterns is limited to the deletion symbol "——————". For addition and replacement, the text passages have to be compared by the amount of new and removed text. If nothing was removed, an *Addition* was detected, if nothing is new except "——————", the *Deletion* is affirmed and else a *Replacement* is found. This combined with the symbolical patterns with the arrows "►" and "▼" give a solid basis for implementing an

[21] `https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:02012R0211-20171211` (Last access date: 09.11.23)

algorithm to find and roughly sort found changes. Taking the arrows into account, the classification ultimately looks like this:

- *Addition* The addition of a block of text or whole article (ADD).
- *Inserted Addition* The smaller addition of a sentence or part of a sentence (ADD).
- *Deletion* The deletion of a block of text or whole article (DELETE).
- *Inserted Deletion* The smaller deletion of a sentence or part of a sentence (DELETE).
- *Replacement* The complete or partial replacement of a block of text or whole article (UPDATE).
- *Inserted Replacement* The smaller replacement of a sentence or part of a sentence (UPDATE).

Patterns for NLP

For knowledge-based Natural Language Processing, there are several kinds of patterns, that can be used. For this use case, an entity analysis might be most useful. Its result highlights entities in sentences, like Organisations, Law references, Countries, Dates, Products and more. This is already shown in [19]. For an analyst reading legal documents, those are the relevant factors that could be filtered further by business management software. For implementation the python library SpaCy[15] is used which already offers entity recognition but also the possibility to extend the patterns used to find those entities. Given that the algorithm only operates with legal documents, phrases like "point(c)" can be added as entity legal reference, thus expanding the accuracy for the recognition of entities. To implement this later, patterns like this have to be created:

```
(1){"label": "LAW",
"pattern": [{"LOWER": "point"}, {"SHAPE": "d", "OP": "+"}]},
(2){"label": "LAW",
"pattern": [{"LOWER": "point"}, {"ORTH": "("}, {}, {"ORTH": ")"}]},
(3){"label": "LAW",
"pattern": [{"TEXT": {"REGEX": "\d{4}/\d{2,4}|\d{2,4}/\d{4}"}}]}, ...
```

Those are examples of the patterns, which SpaCy's entity recognition does not recognize as legal references. Firstly, the patterns, in these cases, are labeled as the entity of a legal reference, so LAW. After that, the real pattern is defined. The first pattern (1) expresses, that one token in lowercase has to equal "point" and the next (nonspace) token after this should be a single digit. This second token has the operator "+", which means this must be there once or more times, like in regular expressions. Thus this pattern recognizes "point1" and "point 13" as LAW. The second pattern (2) again requires one token being "point". After that the first token is supposed to be "(" and the third token ")". The token in between is empty, meaning any token. This makes phrases like "point (a)", "point(1)" or like the earlier example "point(c)" recognizable as a LAW entity. The third example (3) often occurs in EU legislation. A legal document is referenced by its year and number (similar to CELEX numbers) which looks like for example[22] "767/2008" or "2016/399" or even "2017/2226". This is one token that will be matched by a regular expression. Either the part before "/" is the year with 4 digits and the second part includes the number (2 to 4 digits), or the other way around. So the examples above can accurately be labeled as law. Such patterns are only found by running SpaCy with test data to find "unrecognized" entities and adding them by hand as a pattern. This leaves space for improvement and human error but is otherwise not solvable without machine learning. For sufficient accuracy, common knowledge as well as a good amount of test data is needed to find these patterns. Patterns mainly include regular expressions for common phrases including articles, annexes, appendices, paragraphs and points, as well as common notations for EU legal documents and frequently found EU organizations. Overall, there are 4 patterns for organizations (ORG) and 12 patterns for legal references (LAW). There are most probably more to find.

Implementation

This chapter explains the actual implementation of the solution ideas from the previous chapter. To develop the web service, a web framework like flask[23] will be needed as well as a suitable IDE[24]. Bootstrap[25] can be used for designing a suitable frontend. The algorithm, as well as the HTTP connections and the HTML parsing, could best be implemented in Python[26], and additional libraries like requests[27] and BeautifulSoup[28]. For NLP a library like SpaCy[29] could be used, which is used frequently in Related Work. The web service including the algorithm, as well as the data set, are published and accessible in this GitHub[30].

[22]all examples are found in the title of document 02019R0817-20210803
[23]`https://flask.palletsprojects.com` (Last access date: 08.11.23)
[24]In this case PyCharm: `https://www.jetbrains.com/de-de/pycharm/` (Last access date: 08.11.23)
[25]`https://getbootstrap.com` (Last access date: 08.11.23)
[26]`https://www.python.org` (Last access date: 08.11.23)
[27]`https://pypi.org/project/requests/` (Last access date: 08.11.23)
[28]`https://beautiful-soup-4.readthedocs.io/en/latest/` (Last access date: 08.11.23)
[29]`https://spacy.io` (Last access date: 08.11.23)
[30]`https://github.com/affentypi/Webservice_Thesis.git`

Test Data

The data set should work as Golden Standard so the amount of test data should on one hand be representative and on the other still count up to a manageable amount. According to [9] Sector 3 alone includes approximately 22.000 unique legal acts, which again have up to 24 different translations and older versions. The directories of Sector 3 add up to around 27.100 documents in February 2023. For practical purposes, this thesis will concentrate on those documents which are binding legal acts (the mentioned Sector 3) and dismiss non-binding ones, because, in usage, the latter are not as important in their implementation. This means only documents (legal acts) with the descriptors **R** (Regulation), **L** (Directive) and **D** (Decisions) (older decisions are called **S**). Thus the CELEX numbers will look something like "*3(4 digit year)(R/L/D/S)(4 digit identifier)*". The representativity in numbers can not be achieved by hand. A manually harvested set of data has to be selected sophisticatedly to compensate for the missing big data. Thus, the process of finding legal documents started with collecting around one percent of acts from each of the 20 directories[31]. Not every act from these directories has a consolidated version (meaning they have not been changed) which is needed to run as test data, so this needs to be checked before adding them to the collection. In the end, the approximated 1 percent (in February 2023) adds up to 271 legal documents with one or more changes. For every document, the URL addresses of the first released act and the latest updated version were collected as well as an optional "middle" step for legal acts with two or more consolidated versions. These middle steps were selected randomly for the 162 documents that had more than one consolidated version. One document unfortunately was oversized and thus could not be used for testing. This adds up to 270 legal acts with old and new documents that can be compared, as well as 162 of them having a third document, which adds 2 more test possibilities. So there is a total of 594 test that can be made and need expected results counted.

The 'modifications per change' for all detected changes in all 594 document pairs has to be counted. This is done by hand by skipping from modification to modification. This results in numbers looking like "C: x ; M: a + b + c ;" for a document that has one corrigendum with x modifications and 3 amendments with a, b and c modifications, so 4 changes overall. For simplicity, accession treaties are counted as amendments, because they rarely occur and, when they do, they are labeled "Amended by". The "skipping" was manageable because there are links to each "next" modification and it could be checked by searching for the "change-name" as a keyword. Exceptionally, the HTML

[31]`https://eur-lex.europa.eu/browse/directories/legislation.html` (Last access date: 08.11.23)

(mostly old documents) had no links or corrupted links, which made searching for modifications and counting them difficult, but not impossible. However, this leaves room for human error. The resulting numbers are used for a quantitative evaluation checking if the algorithm finds the right amount of changes in the document and the right amount of modifications per change. This can be done by a simple found-to-expected assertion. Additionally, the data set gives examples to debug and to find patterns as explained in Section Patterns for NLP. Making this table, which included the explained steps of collecting URLs, counting modifications and noting patterns, took quite some time. A faster alternative would have been crawling this data with code instead of collecting the URLs, but it would still require counting and noting patterns, so the effort to write an extra code, which again would need checking seemed more time-consuming.

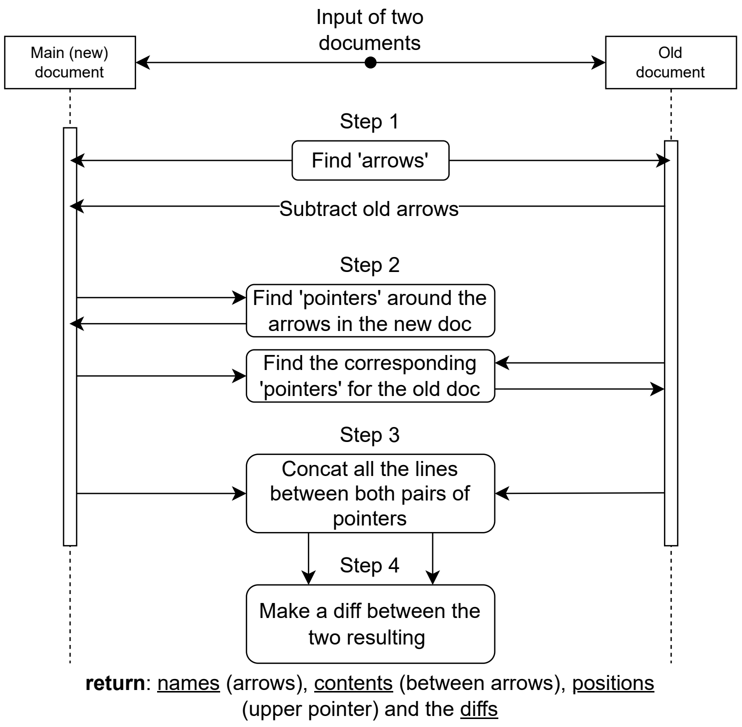
HTML Processing

The legal documents will be input as URLs. The first step of implementation was fetching the HTML text from EURLex[8]. This works through a simple Hypertext Transfer Protocol (HTTP) GET-Request via the Python library requests[27]. In the beginning, the python library urllib3[32] was used, which is sufficient but slower. This switch of library cut the access time roughly in half. The fetched document then is parsed by BeautifulSoup[28], for further processing. Next, parts that differ between the old and new (called main in code) document are to be found. This happens in 4 steps as shown in Figure 6.

1. Firstly, two lists of so-called 'pointers' for each document are created. Those are selected by a regular expression for lines that start with *"Article"*, *"ANNEX"* or *"Appendix"*. The regex is used to avoid mentions and references of "Article xy" throughout the text to be selected as 'pointers'. These 'pointers' outline the structure of the document without relying on line numbers as a reference. The majority of 'pointers' will be found in both lists, with a few exceptions that just occur in either one. Those few will not work for comparison and outlining the document structure.
2. Secondly, the modified parts in the main document are searched. This is done by finding parts in the consolidated version that are marked from amendments ('►M') and corrigendi ('►C') and sometimes accession treaties ('►A'). Those have special HTML markings as well as textual markings in the form of these triangular arrows. The latter is used in this particular instance because it was easier to implement, while still being generally usable. This is

[32]`https://pypi.org/project/urllib3/` (Last access date: 08.11.23)

Figure 6
The four steps of the HTML processing.


Input of two documents
Main (new) document
Old document
Step 1
Find 'arrows'
Subtract old arrows
Step 2
Find 'pointers' around the arrows in the new doc
Find the corresponding 'pointers' for the old doc
Step 3
Concat all the lines between both pairs of pointers
Step 4
Make a diff between the two resulting
return: names (arrows), contents (between arrows), positions (upper pointer) and the diffs

explained in Section Challenges. With the absolute location of the modifications found in the new document, the relative location can be found by searching the nearest surrounding two pointers with a binary search. Those nearest pointers will then be identified in the list of pointers for the old document and accordingly, the modification can be located in the new document as well as where it is supposed to be in the old one.

3. After that, the text between the pointers will be concat, thus resulting in two blocks of text, one old and one new.
4. Lastly, both blocks of texts are compared to result in a so-called diff. Both texts are compared line by line and each line is marked as 'same', 'added' and 'deleted'. This was done using the difflib[33] library. The detour with the pointer saves time and resources by not comparing two full sets of documents but only the modified parts and surrounding context. Also, minor modifications of the text like footnotes that are placed differently do not occur as modifications, because there is no semantic meaning.

[33]https://docs.python.org/3/library/difflib.html (Last access date: 08.11.23)

This all outputs a list of 4 lists. The first list contains the 'change-name' for each modification, the second list the content (the new text that is legally relevant) of each modification and the third the "position", done by taking the upper pointer both documents have in common. The last list is a list of diffs. Those lists are "sorted" by modifications and the indexes align, meaning index 0 of each list concerns the first modification (the change it belongs to, its content, its position and the corresponding diff of its surroundings). The diff compared the text block the modification is in by lines and outputs a list with entries, marked as 'same', 'added' or 'deleted' by the first character being " ", "+" or "-". Unfortunately, the alignment of diff to modification results in duplicate diffs, but the otherwise needed realignment of modification to the corresponding text diff would also take time, not only in cases where multiple modifications are in one diff but in all cases. Also, matching the indicators (name, position, content) to the diff has barriers explained in more detail in Section Challenges.

Natural Language Processing

By inputting a file name, the HTML processing output (a list of 4) and the boolean value for 'fast' (True) or 'accurate' (False) the NLP processing can be executed. Based on the boolean, the SpaCy model[34] is selected: efficiency (sm) or accuracy (trf). Initially, the Blackstone[35] model should be used, but unfortunately, the installation failed due to a missing config file. So the standard web models of SpaCy were used, which already sufficed for other legal-based research like [2] and [10]. The chosen models have no static word vectors because those are only needed for possible future work, as later explained in Chapter Conclusion, and fixating these vectors early would take away the freedom of choice for later work. The alternative to SpaCy[15] would have been NLTK[36] which includes the Standford Parser mentioned in chapter Related Work with the paper [17], but the function-based character of NLTK makes the output less further processable than the object-oriented SpaCy models, which also are more frequently shown in Chapter Related Work. The standard SpaCy models (which were ultimately chosen) are augmented by entity patterns to include Patterns for NLP. The important entities are also colored matching their type and thus also reduced to only the valuable information. Entities like ORG (Organisations), LAW or DATE are important for readers whereas Ordinal or Cardinal markings occur very often with no major semantic value. Information regarding locations like GPE (Countries), NORP (nationalities) and LOC (Locations)

[34]`https://spacy.io/models` (Last access date: 08.11.23)
[35]`https://spacy.io/universe/project/blackstone` (Last access date: 08.11.23)
[36]`https://www.nltk.org` (Last access date: 08.11.23)

have a similar but distinct combination of colors because they contain similar kinds of information. After making these "presets" the output of the HTML processing is further processed. For each modification the *change-name*, *operator*, *position*, *content*, the *nlp-processed content* and a *word-level diff* are appended to the list of modifications for the change the modification is in. This format for each modification is similar to the frame design by [4], but more general, so it fits every "operator". This happens in multiple steps: Firstly, the change list the modification belongs to is found or created, so the 'frame' can be appended at the correct spot. Then the content, which is the text from the newer document is processed by the set-up SpaCy model. When this Bar is found, the operator (meaning the type of modification) can be already identified as Deletion. The position is simply the given from the HTML output. The problem there is, that the "Meta-Data", information about the change at the beginning of the document, is mixed up with modifications in the title or the introduction of a document, as explained in Section Challenges. At this point, the only missing is the diff, the comparison between old and new. Thanks to the alignment diff to modification, the search for the fitting text block is not necessary. Still, to avoid big chunks of text that are output by the line-based diff, a diff on the word level is needed. The line-level diff outputs the lines, the old and new text have in common. From there, the modification is found, simply by the "arrow"s and the change name. The lines around are selected to start and end the comparison with a line that both documents have in common (if possible). Then all the lines are split into words, compared and afterwards, put together according to the new indicators for each word "+" (added), "-" (removed) and " " (same). Thus resulting in a list, where consecutive words with the same indicator are put together while keeping the order of occurrence. If a block of text was updated by only a date, everything will be 'same', except for the added change name plus indicator as well as the exact words that differ. Little mistakes are unfortunately found, for example with syntax like ‘, ´ and ’, which often differ between consolidated versions, and thus are marked as replacements (if inside or near the modification). This is further explained in Section Challenges. Based on the diff, the other operators are evaluated, by comparing how much was added and how much was removed, thus classifying as Addition (if nothing was removed) or as Replacement.

With this list (each index is one change) of lists (all the modifications for this change) an output needs to be made. The function can simply return the values as lists as shown in Figure 7 but to display those, more effort has to be made. The naive approach was to insert the result into an HTML via Script or embedding the results as scalable vector graphics (.svg) into the HTML. However, the

Figure 7

This shows the algorithms output. The first list is a list of all the changes detected. The second is a three-dimensional list, with the first dimension indicating the changes (indexes align with the first list). The second dimension shows given a first-dimensional index (change) a list of modifications that are part of this change. The last dimension is the "frame" of the modifications containing all relevant information about the modification.

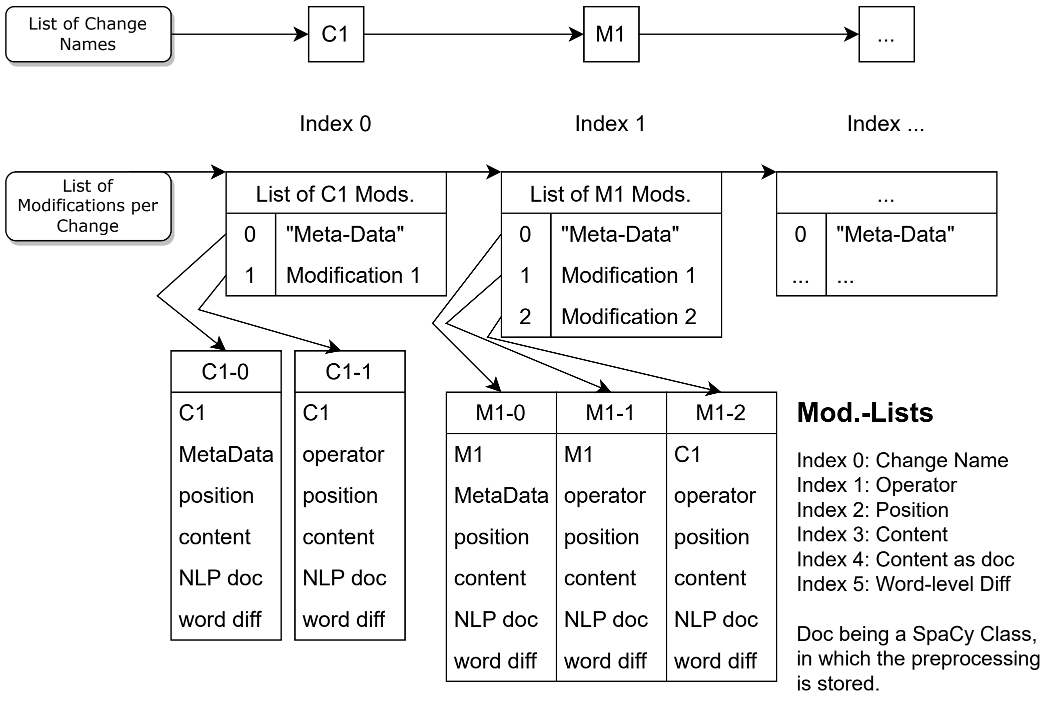
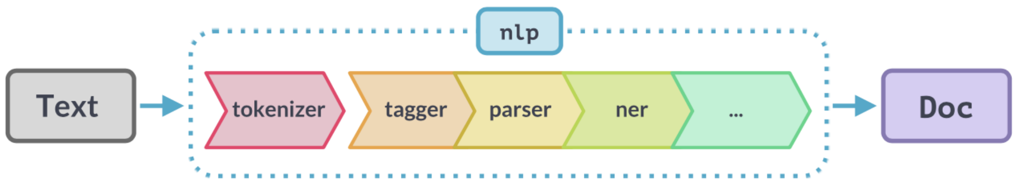


Figure 8

The (shortend) natural language pipeline from `https://spacy.io/usage/processing-pipelines` *(Last access: 07.11.23). Tokenizer, (POS) Tagger, (Dependencies) Parser and (Entity Recognition) NER are shown inside the pipeline. Attribute Ruler and Lemmatizer are missing!*



fact, that 2 variables can not be predicted (the amount of changes and the amount of modification per change) made those approaches more difficult. As an alternative approach, the writing of the HTML file inside the function was used. It needed more ridged work to ensure no syntactic mistakes were made. Overall that approach slows the runtime down, but was an easier way and also easy to remove for future work, that might need more efficiency. The SpaCy visualizer DisplaCy[37] also renders the processed content to an entity highlighted HTML part, which is simply placed in the HTML result. The HTML result is ordered by changes and inside the changes by modifications based on their occurrence. The output done by the function includes the processed content, a SpaCy doc[38]. This means users of the algorithm (not in the web view) can use this output which includes NLP preprocessing and feature generation to work with the contents itself. The doc includes processing by a pipeline (Figure 8) containing Tokenizer (segments the string into words, spaces and punctuation - the most important part!), Part-Of-Speech Tagger (assigns POS tags to tokens like noun, verb, etc.), Dependencies Parser (assigns dependency labels to the tokens like subject, auxiliary verb, etc.), Entity Recognition - with the augmented patterns - (finds entities), Attribute Ruler (handles exception by setting token attributes based on patterns where needed) and Lemmatizer (assigns the token to their base form like "chosen" to the base "choose"). This gives future work the possibility to work with an already processed text, see Chapter Conclusion. Overall, the whole processing is the algorithm with its output being the NLP processing return shown in Figure 7. The web service uses this algorithm as the backend and outputs a user-friendly view of the data, which can not be processed further by a computer without access to the code in the background. Figure 10 in the Appendix shows the whole process from input to output.

Evaluation

The implemented approach was evaluated based on a manually created golden standard, containing 271 EURLex[8] documents. As explained in Section Test Data, for documents with more than one consolidated version, three tests instead of one were performed, which in total resulted in 594 tests. Overall test documents, the right amount of changes and modifications were found in 98.65 percent. The tests were performed with the 'fast' and 'accurate' NLP model and both had the same results of 8

[37]`https://spacy.io/usage/visualizers` (Last access date: 08.11.23)
[38]`https://spacy.io/api/doc` (Last access date: 08.11.23)

Table 2
*A table of all modifications (short mods.) and all changes in one directory for all tests. The names of the directories are shortened to fit the table. 'No.' is the number of tests per directory. The numbers of expected modifications and found modifications are shown. In Directory 3 (marked with *) on document was not tested because of its oversize.*

	Directory	**No.**	**Mods./Changes Expected**	**Found**
General (...) matters	1	42	354 / 65	354 / 65
Customs Union (...)	2	24	493 / 46	493 / 46
Agriculture	3	93	2766 / 401	2766 / 401
Fischeries	4	17	212 / 58	212 / 58
(...) Workers and Social policy	5	18	164 / 39	164 / 39
(...) Services	6	16	194 / 32	194 / 32
Transport policy	**7**	22	***206 / 36***	***204 / 38***
Competition policy	**8**	40	***275 / 76***	***275 / 80***
Taxation	9	7	24 / 9	24 / 9
Economic and Monetary policy (...)	10	13	399 / 31	399 / 31
External Relations	**11**	101	***1159 / 221***	***1178 / 225***
Energy	12	9	61 / 13	61 / 13
Industrial policy (...)	13	48	764 / 112	764 / 112
Regional policy (...)	14	10	527 / 47	527 / 47
Environment	15	75	1240 / 332	1240 / 332
Science (...), Education and Culture	16	13	74 / 14	74 / 14
Law relating to undertakings	17	3	156 / 10	156 / 10
(...) Foreign and Security Policy	18	20	157 / 47	157 / 47
Area of Freedom, Security and Justice	19	20	156 / 31	156 / 31
People's Europe	20	3	3 / 2	3 / 2
Overall	**20**	**594**	**9384 / 1622**	**9401 / 1632**

test files failing. Table 2 shows for each directory[39] how many different documents for each directory are in the data set as well as compares the expected amount of modifications in all these documents to the algorithm found amount of modifications. The numbers are found by running the data set and adding up the found modifications as well as what was expected written in the test data. This saves the process of adding up the expected numbers by hand. The first three documents (not counted in the 271) from the data set were added to debug the algorithm for different languages, which later was not implemented. Those are not used for testing, as was the already mentioned oversized one. The testing consists of 3 tests. The amount of all changes (amendments and corrigendi), the amount of amendments and corrigendi (separate), and the amount of all modifications per document. The latter is summed up for all documents in Table 2. In Table 3 the amount of changes and modifications for all documents are shown, but only from the first released act to the last consolidated version. This reduces the tests to 270 (271 minus oversized). This was to increase the focus on failed tests.

[39]`https://eur-lex.europa.eu/browse/directories/legislation.html` (Last access date: 07.11.23)

Table 3
*A table of modifications and changes found in the comparison of the first act to the latest consolidated version for every document in one directory. No. is the number of tests for each directory which equals the number of documents in each directory. The exception is Directory 3 (marked with *) which has one test less, because of an oversized file that is not tested. Directory 20 (marked with **) has no modifications because it only has one document which was repealed in the last consolidated version.*

Directory	**No.**	**Modifications Expected**	**Found**	**Changes Expected**	**Found**
1	18	191	191	34	34
2	12	245	245	24	24
3	35*	1309	1309	188	188
4	7	108	108	30	30
5	8	84	84	21	21
6	8	120	120	19	19
7	10	***113***	***112***	***20***	***21***
8	20	***148***	***149***	***44***	***47***
9	3	13	13	5	5
10	7	228	228	17	17
11	53	***587***	***606***	***125***	***129***
12	5	32	32	8	8
13	22	392	392	64	64
14	4	257	257	24	24
15	33	565	565	171	171
16	5	41	41	7	7
17	1	80	80	5	5
18	8	80	80	24	24
19	10	80	80	18	18
20	1	0**	0**	0**	0**
Overall	**270**	**4673**	**4692**	**848**	**856**

There are four reasons for a failed test. One of those is **modifications in images** (".jpg" in the case of **32014L0047**[40]; Change M1-3). The file had modifications inside the image and the "linked" modification had footnotes next to the change name. This resulted in two problems: Firstly, the modification could not be identified as part of the given change and "created" a new change. Thus the amount of found changes differed from the amount of expected changes. Secondly, the footnote referred to two modifications but the algorithm can only find this one modification because the text in the image is not readable. The same was found in consolidated versions of **32004R0794**[41] with only one modification. This accounts for 4 of the 8 failures because both had a "middle" document, thus the same mistake was found two times for each act. The other failures were in fairly old documents (2004 and earlier). In **31971R2821**[42] and **31965R0019**[43] **untitled modifications** are found in the text, which both refer by link to the same document 11994N[44] but are not labeled as A3/A4. The algorithm can not make this connection without the change name and 'creates' a new change, thus the result does not equal the expected changes. This is also reflected in the number of modifications because when the test calculates the numbers, for every change one modification (the "Meta-Data") is subtracted, thus resulting in one modification less each. In **31997S1401**[45] a modification of C1 (which has no reference link) is detected but weirdly not assigned to the change C1. This again results in a change more and a modification less. The last, most confusing failure was due to the overuse of indication arrows, seen in Figure 9. In this case, in file 31997S2136[46] the arrow "►" is found before and after the change name M4. The algorithm again thinks this is an unlabeled modification and M4 is empty but both belong to M4 because M4 was the last change of this document and fits the year 2001 column. Those modifications of M4, from the 32001S0244 document was again modified by 32001S2443, but instead of overwriting the M4 modifications, another modification inside the original was created. The test fails because the amount of changes is again different as well as the number of modifications because those unlinked modifications of M3 and M4 failed to be counted for the data set, thus more were found than expected. In total,

[40] `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:02014L0047-20220927` (Last access date: 09.11.23)

[41] `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:02004R0794-20161222` (Last access date: 09.11.23)

[42] `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:01971R2821-20040501` (Last access date: 09.11.23)

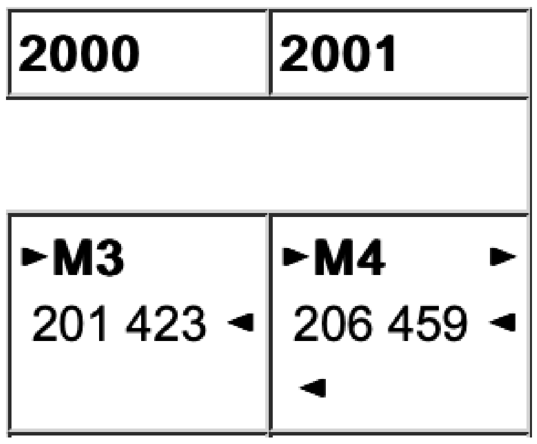
[43] `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:01965R0019-20040501` (Last access date: 09.11.23)

[44] `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:11994N` (Last access date: 09.11.23)

[45] `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:01997S1401-19981004` (Last access date: 09.11.23)

[46] `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:01997S2136-20011215` (Last access date: 09.11.23)

Figure 9
This is an excerpt from the file 01997S2136-20011215, where testing failed to find the modifications. This is found 7 times in this document.



2000	2001
►M3 201 423 ◄	►M4 ► 206 459 ◄ ◄

two fails are from **32014L0047** in directory 7, two from **32004R0794** in directory 8, **31971R2821** and **31965R0019** in directory 8 and **31997S1401** and **31997S2136** in directory 11. The reasons in a nutshell are *modifications in images*, *untitled modifications*, *a recognition error* and *nested overwritten modifications*.

For evaluation, the key figures **Precision** and **Recall** must be calculated. **Precision** is the amount of correctly identified subjects (a subject being a change or a modification) divided by the amount of mistakenly identified subjects plus the amount of correctly identified subjects. **Recall** is the amount of correctly identified subjects divided by the amount of mistakenly not-identified subjects plus the amount of correctly identified subjects. For modifications in Table 3 the amount of correctly identified subjects is **4673**, the amount of mistakenly identified subjects is **20** (**1** in directory 8 and **19** in directory 11) and the amount of mistakenly not-identified subjects is **1** (in directory 7). For changes it is **848**, **8** and **0**. To check these numbers, the overall found should equal the identified added together and subtracting the unidentified (4673 + 20 - 1 = 4692; 848 + 8 - 0 = 856). In Table 2 the numbers for modifications are **9384**, **19** and **2** and for changes **1622**, **10** and **0** (9384 + 19 - 2 = 9401; 1622 + 10 - 0 = 1632).

Overall, the accuracy for a completely correct output of **98.65** percent (586/594) is pretty good but can be improved, by optimizing the algorithm to handle exceptional cases and improving testing by using a different way of counting/comparing the results. Table 4 shows the Precision and Recall values for the two Tables 2 and 3. Table 2 counts all tests, from first to last and if present from

Table 4
Precision and Recall for the values from Table 2 and Table 3.

	Amount of Tests	Precision	Recall
Modifications	270	$\frac{4673}{4673+20} = 0.996$	$\frac{4673}{4673+1} = 0.999$
Changes	270	$\frac{848}{848+8} = 0.991$	$\frac{848}{848+0} = 1.000$
Modifications	594	$\frac{9384}{9384+19} = 0.998$	$\frac{9384}{9384+2} \simeq 1.000$
Changes	594	$\frac{1622}{1622+10} = 0.994$	$\frac{1622}{1622+0} = 1.000$
Average	-	= 0.995	$0.999893 \simeq 1.000$

first to middle and from middle to last. Thus the numbers are far higher. If the wrong amount of changes or modifications is found, this can at maximum happen two times (as for 32014L0047 and 32004R0794) or just one single time, because there is no usable middle document (as for the rest of the failures). This test is still needed if an error occurs only with the middle file. Table 3 only tests first to last, which sets an emphasis on the errors that in the end did occur. Still, the worse mistake of not detecting a modification is less likely than the over-detection of modifications or changes.

On a qualitative basis, the web service is easy to use by supplying the user explanation as well as a simple functionality. To expand the usability, there are ways to only enter the old document (via a link or CELEX number), and the newest (in-force) document is used as a comparison. The performance time is slow, especially slowed by the HTML creation, but acceptable with an average of 7.5 seconds per document or 8 documents per minute (594/73 minutes; 'fast' nlp model). The links to the old and the new document are displayed in the web view, so a user can easily check the whole context of the modifications. The result sorts the modifications by change and in order of appearance, with collapsing boxes, so the user can select, which change to inspect. Each modification consists of the operator (Addition, Deletion, etc.), the position, the content, which is an entity-based NLP-processed view of the new text in the document, and a comparison of old and new text. The entities in the content are colored and labeled. This makes it easy for the reader to focus on important entities like dates or organizations. The entity recognition is limited by the patterns and the standard functionality of SpaCy[15]. But the method of pattern-based Natural Language Processing is more precise than the machine learning approach according to [16] and [6]. The diff displays some context and the actual modification in the text by showing the old part and the new. This diff has some flaws, starting with little syntactic details that are further explained in Section Challenges. But it suffices because the reader can see word for word, what has changed. Some excerpts are found in the git folder test-data[47], where screenshots of the compared passages are shown. The diff depicts the real change but unfortunately, also shows mistakes like modifications from different changes and the same words being marked removed and added. Both mess up the process of classifying modifications, for example, 32013R1308[48] modification M1-1 is supposed to be recognized as an Addition but because of the mixup of as same marked text being mistakenly processed by the diff, the algorithm sees that there is removed and added and classifies the modification as Replacement. The same happens if another replacement or deletion modification is in the diff. A refined diff that does not damage the readability would improve the classification of modifications. Still, an analyst has an overview of all the important information (entities in the new text, comparison of the old and new text, position and assumed type of modification) and the links to retrace the output if needed. The patterns recognize the right entities quite consistently.

In Section Research Questions the research questions are proposed. The first question ("Which patterns are found in changes of legal documents on EURLex[8]?") is evaluated in Section Classification. In short are triangular-shaped "arrows" universally found, where text is modified and the change this modification belongs to is next to this "arrow". To answer the second and third questions, ("How can these patterns be classified and used to support information extraction?" and "What NLP techniques or approaches are most suitable to extract data from changed text?") the patterns from Chapter Solution Design are looked at. The arrow symbols from Section Classification are useful for finding and working with the modifications and connecting them to the right change. The patterns from Section Patterns for NLP are useful to extract the relevant information from the modifications, in this case, the entities. The approach is easily expandable, but leaves room for errors. The output however does not only include these entities but a SpaCy[15] doc type. This includes the whole pipeline of NLP preprocessing and NLP feature generation as explained in Section Information Extraction by Natural Language Processing with the paper [16]. So the results can be further processed to analyze sentence structure, filter for specifics or implement a machine-learning model. This is not included in the web service, because the output is supposed to be generally usable and those further processes are supposed to pursue a scientific or business-relevant issue, which this paper does not have. Also, it gives the user the possibility to use this algorithm as a starting point for more in-depth work. The last question ("How can changes be displayed to aid hybrid systems for legal business compliance?")

[47]The selection of comparisons in the folder is quite small because fitting all the passagex (result from the web service, old text and new text) in one overview is only possible with rather small modifications.

[48]`https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:02013R1308-20230101&qid=1699627292726` (Last access date: 10.11.23)

consists of two answers: For the machine part of the "display" or better output, the changes are output in the form of a list of the modifications they consist of (Fig. 7). These modifications further consist of all the relevant data: the name of the change, the position of the modification, the operator (which class of modification) the new content in processed form and a "differences" list of old vs new. This combines the simple (rigid) tuple format from [16] with the FRAME output from [4]. The rigid format makes an easy iteration possible because the position of each piece of information is fixed to an index in the list for each modification. The visible output for a user of the web service shows the changes as expandable blocks, which when expanded show all the modifications with highlighted entities as well as a comparison of the old and new passages. The produced artifacts (as demanded by the Research Methodology) are the data set of 271 individual documents (*instantiation*), the patterns and classification from Chapter Solution Design (*model*) and the web service and its Implementation (*method*).

Discussion

Contribution

This thesis' contribution is classifying changes to legal documents from the European Union and finding an efficient way of extracting information regarding those changes. A state-of-the-art workflow would be finding a particular change/modification in the new document, opening the old document, searching for this particular text passage and comparing, what was changed. This includes multiple steps; especially the search for the right text passage in the old document can be time-consuming. Furthermore, for thorough work, this has to be repeated for every modification in the new document. This web service is supposed to make these steps for all modifications. Besides this, the web service highlights key entities in the newly changed legal text that are commonly of interest, like organizations, dates and references to other Legal documents, while also outputting the differences between old and new. A possible workflow can be, that an analyst enters the last used legal document for a check-up (and optional the latest or some newer state of this legal document) and the tool shows, what changes occurred since the last check-up, which is presented readable to the analyst but also as a Python list, which can be further processed by programs. This way, a human can still comprehend the changes and a program, like a process compliance management system can adopt the legal changes. This process is shown in Figure 1 in Section Motivation.

Challenges

The first challenges came in HTML Processing. After receiving the HTML, the question was whether to extract the text and only work with the text or also use the HTML tags and their information. The latter offers more insights but sadly comes with several hurdles. The first and most important is, that the text containing HTML parts has changed over the years. Many HTML documents used the class "normal", while some used "norm". And this extended every other file, where files used the class "text", class "content" or simply "<p>" or "<div>" paragraphs. Finding a solution using HTML that works on all files seemed impossible. Furthermore, the HTML tags, that were interesting for processing (like links to modifications), are only one small part of a big list, whose order again varied from file to file. The simpler textual processing was more reliable, especially because the arrow markings (explained in Section Classification and Section HTML Processing) were found to be used unanimously. So plain text was used, dismissing the information stored in the HTML.

The next problem resulted from an attempt to make the code more efficient. The output explained in Section HTML Processing initially should not have the diffs aligned with the modifications. The idea was not to process the same text again if it had already been processed before. But this left the problem, that every modification had to iterate through all diffs to find the right one. The problem with that approach was that the indicators available (the name of the change, the content of the modification and the position) did not suffice to completely find the right diff or the right passage in the diff. For example, a simple deletion could only be found in the old document by the position which leaves the whole Article or ANNEX. This would be still manageable for only one modification in this text block, but as soon as there are replacements or additions, the whole process gets complicated by finding out which modification is which. One attempt to solve this problem was comparing all the sentences in the text block, which caused more chaos. A simple textual comparison did not work because the formatting of the text mostly differs. Examples would be the syntactic difference between ´, ‘ and ’ or simply the spaces between words. To handle this, the idea was to use the cosine similarity of whole sentences to cancel out those little differences. It looked promising at first even though the performance suffered. This stopped working for diffs, where the lines do not represent the sentences. Often Sentences would be split over multiple lines in one document and be in one single line in the other for seemingly no reason and the results of the comparison were not usable. Also, one document had moved its footnotes from inside the article (32019R0817[49]) to the end of the document (02019R0817-20210803[50]), which added many "removed" parts to a simple replacement. There might be more examples of those problems but at this point, this approach was discarded, because there was no guarantee this way would work universally. The solution was, to align the diff to the corresponding modification, so the search of which diff concerned which modification was not necessary anymore.

This left one problem in the room: Many diffs contained in their "added" and "deleted" sections the same part sentences or even the whole sentences. This problem had to be solved by a diff on word level like explained in Section Natural Language Processing. This needed much attention because it was important to keep the order of lines and words intact while only selecting the few ones that were the context of the modification. This works okay, but if multiple modifications with the same change name appear in the same text block, all of them will have the same diff. Also, the word diff still can not cope with little syntax signs like ´, ‘ and ’, so phrases in the context or in the modification with differences on this level are also marked as "removed" and "added". Many different approaches to refine the diff and make it more accurate failed because as soon as the accuracy of detecting the real modification got higher, the usability and readability of the web service output decreased. But because the classification is dependent on it, it was a high priority. Fixing the diff took approximately 30 percent of the coding time for the algorithm. The current state still could be refined.

Lastly, a big challenge was to find every type of exception there could be. This was mainly done by thorough work in the obtaining of documents for the data set, where many examples could be found and noted for the implementation. Sadly, in Chapter Evaluation, some "mistakes" could be found and of course, there is a probability, that some edge cases are simply not present in the data set. One of those is modifications in the title or the introduction as mentioned in Section Natural Language Processing. Both are identified by the algorithm as "Meta-Data". The best example is 02004D0452-20120420 with the corrigendum C1. Both the title and introduction were modified and in between is the name, date and link to the amending act (so-called "Meta-Data"). The first idea to sort this out was to use the URL of the arrows, where modifications are assigned numbers to navigate the document. Those are appended at the end of the URL as "C1-1". But EURLex[8] missed "numbering" the title and just labeled the "Meta-Data" as C1-1 instead of the title modification. The

[49]`https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32019R0817` (Last access date: 09.11.23)

[50]`https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:02019R0817-20210803` (Last access date: 09.11.23)

simple idea of "just take the second Meta-Data as Meta-Data" does not work because while most of these cases only have a modification in the title, some have only a modification in the introduction, so again it would not be a solution that would work generally. The last and also fruitless attempt was by the content, but the title/intro and "Meta-Data" often are pretty similar because they both state the type of legal act (Decision, Regulation and so on) and the date. The only difference is the actual document name. The names in titles and intros of course contain many unpredictable words but a few words like "laying down", "applying" and "concerning" are pretty common. But again there are many exceptions with completely different names, so it would only work for a "dictionary" of keywords that may or may not occur. The consequence of this unsolved problem is the mislabeling of modifications as "Meta-Data". Furthermore, exceptions are found in documents created in 2004 and earlier which were formatted in a completely different way. The only exception identified and implemented is when documents are written in a single paragraph with only spaces between titles, articles, paragraphs and footnotes. This was solved by an extra step splitting the text into lines containing sentences until the lines have a similar form as the current documents have. Other not implemented exceptions can be found in the failed tests from Evaluation. But there is a high probability that these old documents have further exceptions which are not found yet.

Conclusion

This thesis developed an automated way to find changes and modifications in legal documents and extract information from them. The web service its algorithm output with an accuracy of 98.65 percent on 584 tests all modifications in legal documents comparing them to older versions. The thesis plan, shown in Figure 5, consisted of identifying and classifying patterns as explained in Chapter Solution Design and developing a tool (the web service) as shown in Chapter Implementation. Furthermore, to find patterns and test the tool, a data set was created which size assured a wide range of documents as well as thorough testing as seen in Chapter Evaluation. The Motivation was to support the process of keeping up with a changing legal landscape without long iterative work. This is possible by using the web service. Instead of navigating and comparing documents by hand, the documents can be processed by the web service and will output the modifications clearly arranged and human-readable, thus reducing the work of researching. For a fully automated process, the algorithm for the web service can be integrated into a business compliance software to update the regulatory documents according to its changing legal basis.

For future work, many possibilities are shown to be open. First of all, the web service could be deployed to be universally accessible. Furthermore, tweaking and corrections could be made to the algorithm and web service. The diff is definitely one part of the output, that could be better displayed as well as better processed. The processing could be made smoother with less common text in old and new, which would make identifying the actual change easier. Also, the performance could be optimized, by using more efficient methods and algorithms (like binary search as used once) inside the algorithm. This would be needed for more time-sensitive, big-data use cases. Another step would be, instead of using pattern-based NLP, training a model to create a more generalized entity recognition. But that makes only sense if the data set would be expanded by written results with by-hand-processed information to have statistical training data. This paper did not go this far, based on the finite amount of entity patterns, as well as the recommendation by [6]. For the web service safety and security measures could be implemented, to prevent users from accidentally or purposefully misusing the web service.

Aside from optimizing the existing, the algorithm's output can be the basis for further research or in-depth work. For example, the output includes for each token the labels of dependencies, so the sentence structure could be classified and used for further research like [10] and [12] did with legal sentences in German. This could be further extended by making Abstract Syntax Trees (AST) like [14] using the dependency labels. Another example would be creating a machine-learning model. This model would need a specific issue, either for research or business applications, like categorizing the importance (like [8]) or filters for specific entities or keywords. The data set from Test Data could be used in the first step as "learning material". To assure freedom in further steps, the output has no word vectors, so future work can select the fitting vectors and algorithms like SVM from the paper [7] as used in [2]. Furthermore, the data set used for testing could be used in other use cases working with legal documents from EURLex[8].

In conclusion, this paper presented generalized patterns to find and extract changes and their modifications from legal documents from EURLex[8] as well as implemented a web service based on an algorithm to output these modifications processed by pattern-based Natural Language Processing for humans (web service's output) and machines (algorithm's output). The data set used as the golden standard for testing is quite big with 271 documents and 594 possible tests. On this basis, the accuracy of 98.65 percent for a completely correct output and the Precision and Recall values with none under 0.990 for detecting changes and their modifications are good.

Bibliography

[1] M. Hashmi, G. Governatori, H.-P. Lam, and M. T. Wynn, "Are we done with business process compliance: State of the art and challenges ahead," *Knowledge and Information Systems*, vol. 57, no. 1, pp. 79–133, ISSN: 0219-3116. DOI: 10.1007/s10115-017-1142-1. [Online]. Available: https://doi.org/10.1007/s10115-017-1142-1.

[2] S. Fabrizi, M. Iacono, A. Tesei, and L. De Mattei, "A first step towards automatic consolidation of legal acts: Reliable classification of textual modifications," in *Proceedings of the Eighth Italian Conference on Computational Linguistics CliC-it 2021*, E. Fersini, M. Passarotti, and V. Patti, Eds., Accademia University Press, pp. 141–147, ISBN: 9791280136947. DOI: 10.4000/books.aaccademia.10619. [Online]. Available: http://books.openedition.org/aaccademia/10619.

[3] L. Lesmo, A. Mazzei, M. Palmirani, and D. Radicioni, "TULSI: An NLP system for extracting legal modificatory provisions," *Artificial Intelligence and Law*, vol. 21, DOI: 10.1007/s10506-012-9127-6.

[4] C. Biagioli, E. Francesconi, A. Passerini, S. Montemagni, and C. Soria, "Automatic semantics extraction in law documents," pp. 133–140. DOI: 10.1145/1165485.1165506.

[5] A. Hevner, A. R, S. March, *et al.*, "Design science in information systems research," *Management Information Systems Quarterly*, vol. 28, p. 75,

[6] E. de Maat, K. Krabben, and R. Winkels, "Machine learning versus knowledge based classification of legal texts," *Legal Knowledge and Information Systems*, pp. 87–96, Publisher: IOS Press. DOI: 10.3233/978-1-60750-682-9-87. [Online]. Available: https://ebooks.iospress.nl/doi/10.3233/978-1-60750-682-9-87.

[7] C. Cortes and V. Vapnik, "Support-vector network," *Machine Learning*, vol. 20, no. 3, pp. 273–297, ISSN: 08856125. DOI: 10.1023/A:1022627411411. [Online]. Available: http://link.springer.com/10.1023/A:1022627411411.

[8] K. Asooja, O. Ó. Foghlú, B. Ó. Domhnaill, G. Marchin, and S. McGrath, "Automatic detection of significant updates in regulatory documents," in *Legal Knowledge and Information Systems - JURIX 2017: The Thirtieth Annual Conference, Luxembourg, 13-15 December 2017*, A. Z. Wyner and G. Casini, Eds., ser. Frontiers in Artificial Intelligence and Applications, vol. 302, IOS Press, pp. 165–169. DOI: 10.3233/978-1-61499-838-9-165.

[9] D. Aumiller, A. Chouhan, and M. Gertz, *EUR-lex-sum: A multi- and cross-lingual dataset for long-form summarization in the legal domain.* DOI: `10.48550/arXiv.2210.13448`. arXiv: `2210.13448[cs]`. [Online]. Available: `http://arxiv.org/abs/2210.13448`.

[10] I. Glaser, E. Scepankova, and F. Matthes, "Classifying semantic types of legal sentences: Portability of machine learning models," *Legal Knowledge and Information Systems*, pp. 61–70, Publisher: IOS Press. DOI: `10.3233/978-1-61499-935-5-61`. [Online]. Available: `https://ebooks.iospress.nl/doi/10.3233/978-1-61499-935-5-61`.

[11] H. J. Meijer, J. Truong, and R. Karimi, *Document embedding for scientific articles: Efficacy of word embeddings vs TFIDF.* DOI: `10.48550/arXiv.2107.05151`. arXiv: `2107.05151[cs]`. [Online]. Available: `http://arxiv.org/abs/2107.05151`.

[12] B. Waltl, J. Muhr, I. Glaser, G. Bonczek, E. Scepankova, and F. Matthes, "Classifying legal norms with active machine learning."

[13] R. Bartolini, A. Lenci, S. Montemagni, V. Pirrelli, and C. Soria, "Semantic mark-up of italian legal texts through NLP-based techniques,"

[14] N. Dinesh, A. Joshi, I. Lee, and B. Webber, "Extracting formal specifications from natural language regulatory documents," in *Proceedings of the Fifth International Workshop on Inference in Computational Semantics (ICoS-5).* [Online]. Available: `https://aclanthology.org/W06-3902`.

[15] A. Ranta, I. Listenmaa, J. Soh, and M. W. Wong, "An end-to-end pipeline from law text to logical formulas," *Legal Knowledge and Information Systems*, pp. 237–242, Publisher: IOS Press. DOI: `10.3233/FAIA220473`. [Online]. Available: `https://ebooks.iospress.nl/doi/10.3233/FAIA220473`.

[16] J. Zhang and N. M. El-Gohary, "Semantic NLP-based information extraction from construction regulatory documents for automated compliance checking," *Journal of Computing in Civil Engineering*, vol. 30, no. 2, p. 04015014, ISSN: 0887-3801, 1943-5487. DOI: `10.1061/(ASCE)CP.1943-5487.0000346`. [Online]. Available: `https://ascelibrary.org/doi/10.1061/%28ASCE%29CP.1943-5487.0000346`.

[17] M. Dragoni, S. Villata, W. Rizzi, and G. Governatori, “Combining NLP approaches for rule extraction from legal documents,” presented at the 1st Workshop on MIning and REasoning with Legal texts (MIREL 2016). [Online]. Available: https://hal.science/hal-01572443.

[18] A. Korger and J. Baumeister, “Rule-based semantic relation extraction in regulatory documents,”

[19] P. Quaresma and T. Gonçalves, “Using linguistic information and machine learning techniques to identify entities from juridical documents,” in *Semantic Processing of Legal Texts: Where the Language of Law Meets the Law of Language*, ser. Lecture Notes in Computer Science, E. Francesconi, S. Montemagni, W. Peters, and D. Tiscornia, Eds., Berlin, Heidelberg: Springer, pp. 44–59, ISBN: 978-3-642-12837-0. DOI: 10.1007/978-3-642-12837-0_3. [Online]. Available: https://doi.org/10.1007/978-3-642-12837-0_3.

[20] T. H. Beach, Y. Rezgui, H. Li, and T. Kasim, “A rule-based semantic approach for automated regulatory compliance in the construction sector,” *Expert Systems with Applications*, vol. 42, no. 12, pp. 5219–5231, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2015.02.029. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417415001360.

[21] E. Hjelseth and N. Nisbet, “EXPLORING SEMANTIC BASED MODEL CHECKING,”

[22] H. Hapke and H. Lane, *Natural Language Processing in Action*, ISBN: 978-1-61729-463-1. [Online]. Available: https://learning.oreilly.com/library/view/natural-language-processing/9781617294631/.

Appendix

Terminology

- Abstract Syntax Tree (AST): As used in [14] and [15], ASTs represent the syntactical structure of a text as a tree. The nodes contain logical constructs from the text written in a formal language.
- CELEX[51]: A unique identifier of each document in EURLex[8], regardless of which language it is in.
- Change vs. Modification: A change is an amendment or corrigendum put in effect by lawmakers and consists of one or more modifications, which are the real textual alterations.
- Frame: A way of representing information via a FRAME, consisting of provision class and slots of class-specific legal entities. An example of provision classes and their slots is seen in Fig. 4.
- Index: The position in a list, starting with 0 for the first place.
- Legal Texts: Texts formulated and entered into force by lawmakers as legal acts.
- NLP: Natural Language Processing is the technique to make computers 'understand' and process natural language, written by humans.
- n-gram: The idea of n-grams is to separate texts into small linguistic items in a contiguous sequence and count the repetitions. The size of n denotes how many sequential items are listed, n = 1 is called unigram.
- Part-of-Speech (POS) tagging: The process of marking words in a text by their role, like subject, object, verb, and so on.
- Regulatory Documents: Formal guidelines to apply the principles of acts. Breaching them is not necessarily enforceable in courts.
- SVM: Support Vector Machine is a machine learning algorithm introduced in [7].
- Tokenization: A process in lexical analysis where strings are split into items, that could be used for processing like for example splitting a string into words.

[51]`https://eur-lex.europa.eu/content/help/eurlex-content/celex-number.html` (Last access date: 07.11.23)

Figure 10

This sequence diagram shows the steps of the web service. This is the refined process of Figure 2. The "Return lists" are shown in Figure 7.

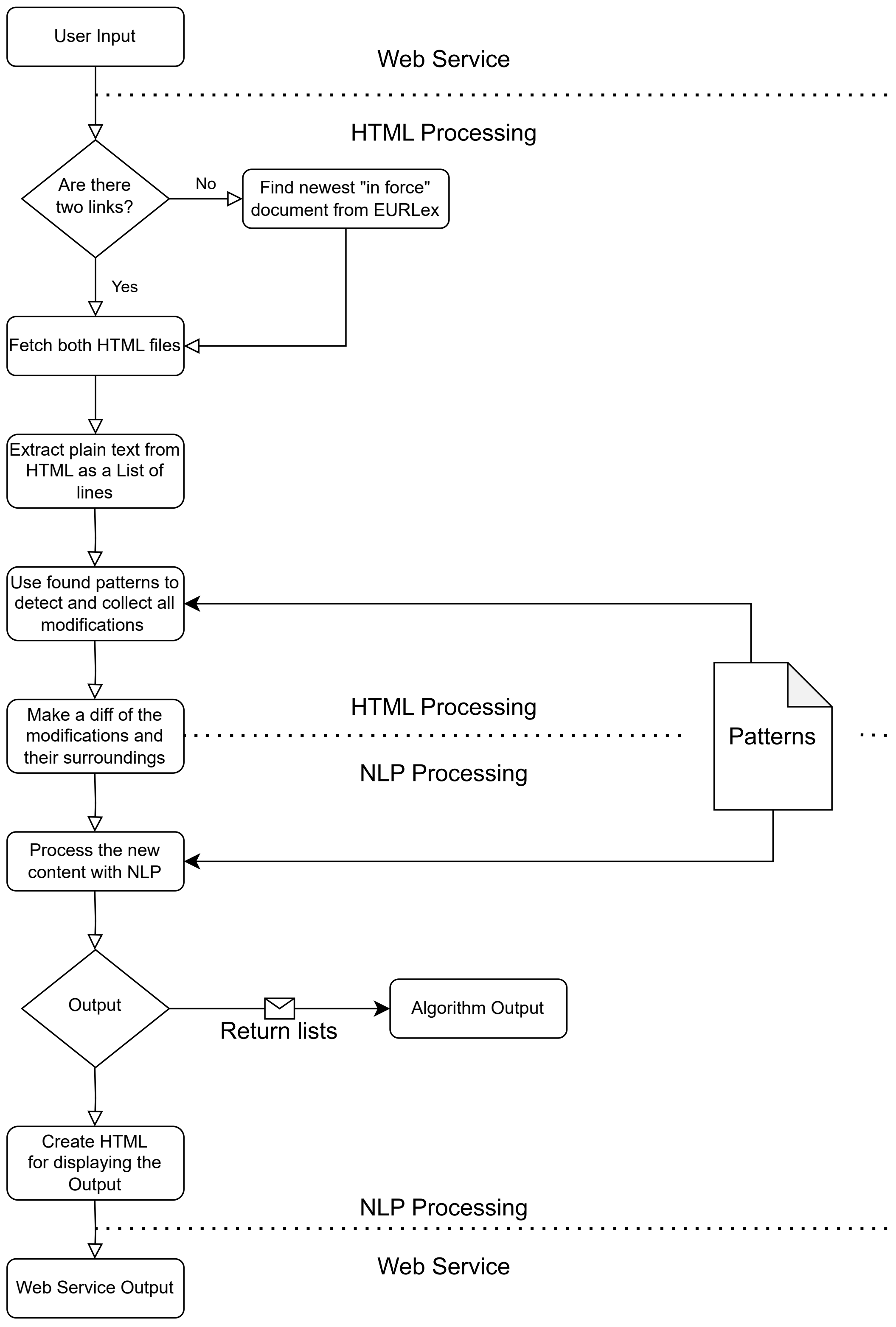

User Input
Web Service
HTML Processing
Are there two links?
No
Find newest "in force" document from EURLex
Yes
Fetch both HTML files
Extract plain text from HTML as a List of lines
Use found patterns to detect and collect all modifications
Make a diff of the modifications and their surroundings
HTML Processing
NLP Processing
Patterns
Process the new content with NLP
Output
Return lists
Algorithm Output
Create HTML for displaying the Output
NLP Processing
Web Service
Web Service Output

Table 5

The words counted by `wordcount.com` *from each HTML-formatted amending legislative act of 2022. The letters in the front collum identify the acts as Regulations (R), Directives (L) or Decisions (D). The acts are identified by the last 4 digits of their CELEX number.*

	Act	Words	Characters	Sentences
32022R	2495	1595	15478	56
32022R	2480	1136	9939	36
32022R	2434	1346	11068	45
32022R	2400	2644	22848	50
32022R	2370	11879	94753	288
32022R	2036	4862	38419	99
32022R	2039	3635	28518	81
32022R	2040	1118	8963	40
32022R	2038	4825	35053	101
32022R	2037	1340	10607	26
32022R	1278	2003	15467	50
32022R	1190	4567	34531	112
32022R	1034	4515	34565	110
32022R	1035	1604	13958	42
32022R	1032	8775	65990	231
32022R	1033	1237	9586	35
32022R	0991	23144	170934	539
32022R	0992	563	4483	19
32022R	0838	2182	16932	57
32022R	0641	1019	8188	30
32022R	0613	1894	14863	40
32022R	0590	7117	58494	371
32022R	0585	2621	20199	62
32022R	0562	2574	19925	66
32022R	0312	1175	9026	33
32022R	0112	1901	1466	44
32022R	0111	401	3440	14
32022L	2556	4640	38935	86
32022L	2464	34014	272406	806
32022L	2380	6937	55230	178
32022L	0738	2810	21691	85
32022L	0642	5099	38192	105
32022L	0431	5234	41666	151
32022L	0362	17438	133915	445
32022L	0228	782	6264	26
32022L	0211	809	646	26
32022D	0871	862	6984	24
	Max	34014	272406	806
	Min	401	646	14
	Avg	4872,8	37665,4	124,5