Final Report for Masterpraktikum:
Approaching Information Systems Challenges with NLP

Munich, 29.01.2024

By Xena Striebel

Student ID: 03781135

1. Introduction

English language in professional and academic realms commonly uses passive constructions to express neutrality and describe procedures. Especially in legal documents, passive voice often adds another level of complexity to already long and nested sentences. Combined with legal terminology, this results in legal texts being very abstract and neither understandable nor intuitive for the majority of non-experts. One option to reduce complexity and make legal documents more accessible for laypeople is to transform passive sentences into active voice. Performing this procedure manually would be very tedious and inefficient. Instead, it would be useful to use a computer algorithm or similar to do it automatically. For this use case, Natural Language Processing (NLP) can be applied. NLP "is a field of computer science, and artificial intelligence and scientific study of language that deals with the interactions between computers and human Languages" (Agrawal et al., 2016). However, to the best of my knowledge, there exists neither a library nor another program using a rule-based approach which is already capable of transforming complex, long legal, and regulatory passive sentences into active voice correctly. Therefore, I implemented a rule-based algorithm in Python to approach this problem. In the next paragraphs, the basic procedure of developing the algorithm, the arisen challenges, and specific approaches, as well as the evaluation and limitations of the algorithm, are elaborated.

2. Preparation

2.1. Golden standard

Before the main algorithm was implemented, a golden standard was created to have sentences and their reference sentences to evaluate the final algorithm with. Herein, 160 sentences with at least one passive construction were extracted from the German basic law, the GDPR, and the medical device requirements from the EU. For each of the sentences, the same sentence in active voice was defined as the expected result, to later be able to compare the output of the algorithm with the expected output. In order to obtain a collection of sentences of varying length and complexity, the passive sentences were selected according to the following criteria:

1. Diverse length
2. Diverse complexity and structure
3. Different tenses
4. Sentences with relative clauses and subclauses
5. Some sentences with negations
6. Some sentences with more than one passive construction

The active sentences were mainly produced manually and checked for grammatical errors or misspellings via Grammarly and another human. Even though the production of the active sentences was done very carefully, during the process of implementing the algorithm, it was noticed that some of the sentences had some minor errors like a missing comma or instead of a should as modal verb a shall. To be able to do the evaluation correctly, these minor errors were adjusted in the Golden standard and marked by colouring the relevant row in yellow and highlighting the change (see file Goldstandard_updated.xlsx). Therefore, the adjustments should be very transparent.

2.2. Related Work

Several studies have explored basic methods for converting sentences between active and passive voice, each with its own limitations. Fahad and Beenish (2020) developed a prototype using context-free grammar (CFG) to convert active sentences into passive, but it's effective only for simple sentences with a basic structure of subject, verb, and object. Agrawal et al. (2016) also created an algorithm for transforming active sentences into passive, with similar limitations in handling complexity. In a different language context, Ilukkumbura and Rupasinghe (2021) applied a rule-based approach to convert active voice into passive voice in Sinhala. However, there the focus is on three-word sentences and, thus, is restricted to very simple sentence structures. Nohimovich and Jin

(2018) implemented a rule-based pipeline for such transformations, but it is limited to specific cases and not applicable for varied sentence structures. Pawale et al. (2015) took a broader approach, employing syntax parsing, CFG, and part-of-speech tagging for correcting sentences in English, indicating a move towards handling more complex sentence structures. Moreover, Tayal et al. (2014) introduced a rule-based method to assess syntactic correctness using pre-defined CFG rules, further highlighting the challenges in dealing with complex sentences.

In conclusion, these works - while proposing suitable approaches - are mostly confined to predefined, simple sentence structures and tackle the conversion between passive and active voice at a basic level. This underscores the complexity of creating a more versatile and comprehensive system capable of handling a wide array of sentence structures. The algorithm presented in this paper builds upon these foundational works, incorporating new ideas and methods to extend their capabilities.

3.Project Implementation Procedure

3.1. Basic implementation pipeline

Based on the aforementioned related work and combined with own ideas and approaches, the following basic pipeline for the algorithm was developed:

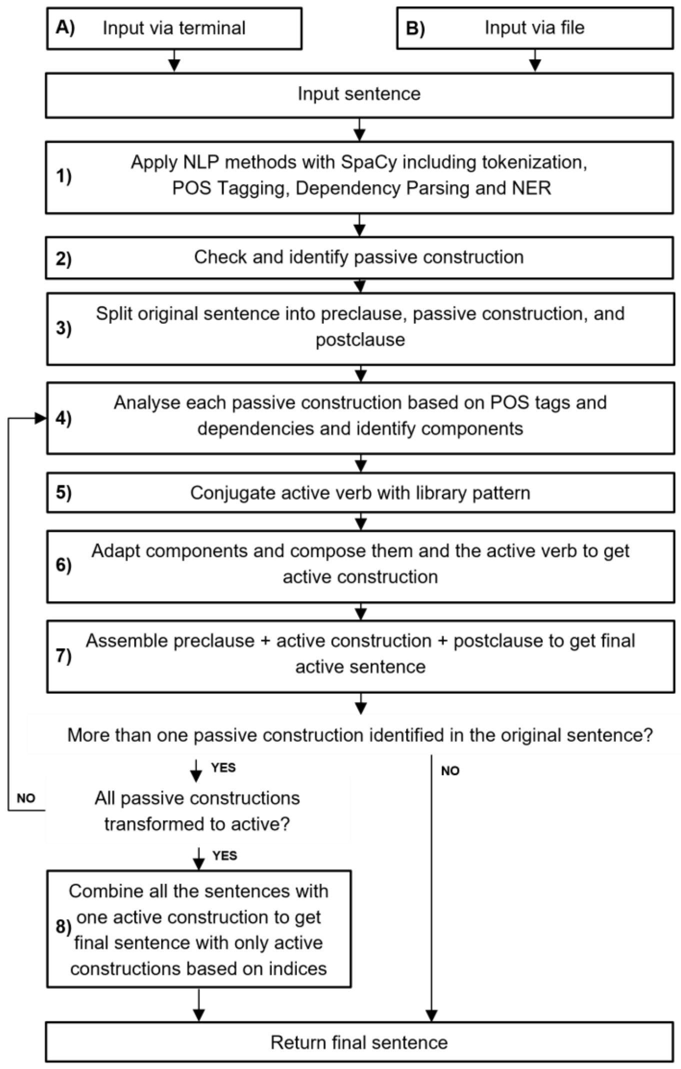
A) Input via terminal
B) Input via file
Input sentence
1) Apply NLP methods with SpaCy including tokenization, POS Tagging, Dependency Parsing and NER
2) Check and identify passive construction
3) Split original sentence into preclause, passive construction, and postclause
4) Analyse each passive construction based on POS tags and dependencies and identify components
5) Conjugate active verb with library pattern
6) Adapt components and compose them and the active verb to get active construction
7) Assemble preclause + active construction + postclause to get final active sentence
More than one passive construction identified in the original sentence?
YES
NO
All passive constructions transformed to active?
NO
YES
8) Combine all the sentences with one active construction to get final sentence with only active constructions based on indices
Return final sentence

Figure 1: basic implementation pipeline

The algorithm can be used either to just convert one sentence or to convert a whole file with passive sentences. The requirements for the file so that it can be dealt with appropriately are:

- xlsx-File
- One column called "InputSentence" with the sentences which should be converted, one in each row
- The path of the file can be inserted during running time in the terminal
- In order that the transformation can be evaluated automatically, there also needs to be a column
    - Called "ReferenceSentence", which inherits the expected output, i.e. the active sentence and one column
    - Called "Mode" where it is written whether a sentence is passive or active

In both scenarios, A and B, one sentence at a time is taken, and the main function passiveToActive is applied. In this function, all the analysis, transformation, and composition of the final sentence is done.

Step 1: First the sentence is processed by performing some NLP tasks like Tokenization, POS Tagging, Dependency Parsing and Named Entity Recognition (NER). For that, the library spaCy is used because of its advanced linguistic features, high performance, and ease of integration in Python applications. For all the NLP tasks, the large English language model ('en_core_web_lg') is used, which provides high accuracy, robustness, and contextual understanding since it is trained on a wide range of text data, making it very versatile and accurate.

Step 2: Then, it is assessed whether the current sentence is in passive voice by checking whether the sentence contains a word with the dependency *nsubjpass* or *csubjpass* and one with the dependency *auxpass*. *Nsubjpass*/*csubjpass* indicates that this word is a nominal/clausal subject in a passive construction, and auxpass stands for all forms of "to be" in passive constructions, e.g. *was* written. Thus, by searching for these two dependencies, passive constructions can be identified reliably. If both are identified and part of the same structure, the subclause of the head of the *auxpass* is extracted from the original sentence and stored as the passive construction.

Step 3: If the identified construction is not the main clause, the parts before/after this subclause are stored as *preclause/postclause*. Obviously, they can be empty if sentences are very short and only consist of one main sentence having one passive construction. This step is done for so long as all passive constructions in the original sentences are identified, meaning that if there is more than one passive construction e.g. in the main clause and subclauses, they and their corresponding *pre-* and *postclauses* are stored. Also, the position of the passive construction in the original sentence is stored by extracting the indices. This is done, to being able to compose the final sentence later. If no passive construction is identified, the process is stopped, and it is waited until the next sentence is entered. Otherwise, the identified passive constructions, together with *their preclause, postclause,* and indices, are returned.

Step 4: Next, every identified passive construction and its morphological characteristics are analysed by using the results of the earlier performed NLP tasks. Then, each word of the construction is analysed to determine whether it has an important transformation role in the sentence. This can be, for instance, next to nsubjpass and auxpass, also the root as predicate, dependencies that indicate subclauses like *acl, advcl, relcl, xcomp, ccomp*, or adverbial, adjectival, or prepositional modifier and others. The goal here is to get all parts of the sentence and assign either the word or, if existent, its subtree to a variable. Also, for the later conjugation of the verb, the verb characteristics like time, number, mode, modal verbs, etc, are stored from the *auxpass*. After all words have been analysed, all the variables and their values are returned.

Step 5: Then, considering the previously identified tense, mode, number etc., the active verb form is constructed with its necessary modal and auxiliary verbs. For the conjugation, the function *conjugate()* from the library *pattern* is used. In this step, it was rather difficult to cover all possible constructions of verbs, like, for instance, third-person adaptions, negation, or the conjugation of have if it`s used as an auxiliary verb in forming tenses.

Step 6: After that, if necessary, the subject and object are inversed accordingly. Herein, if there is no agent provided in the sentence, the algorithm uses “one” as the default subject for the active sentence. The results from the sentence analysis, including agent, subclauses, prepositions, etc and the active verb, are used to compose the active construction.

Step 7: Then, the active construction is put between the earlier identified *preclause* and *postclause* so that finally, the original sentence is adjusted in such a way that the passive construction is replaced with the just-produced active construction.

If there has only been one passive construction in the original sentence, the process is over, and the final sentence is returned. Otherwise, for every other passive construction, the same procedure as described is done. Then, to be clear, there are as many new sentences as how many passive constructions have been identified because, as described, every identified passive construction is replaced in the original sentence, and for that, we have a separate (sub)sentence.

Step 8: After that, since earlier the indices of the position of the passive construction in the original sentence were stored, now they are used to add all the sentences with one transformed active construction to get one sentence with all transformed active constructions.

Table 1 illustrates the procedure of a sentence being split into *preclause*, passive construction and *postclause:*

Original sentence:	These details, which shall be regulated by a federal law, require consent of the Bundestag.		
Split into its parts (Step 3):	Preclause	Passive construction	Postclause
	These details,	which **shall be regulated by a federal law,**	require consent of the Bundestag.
Transformation of only the passive construction (Step 4 to 6):		which **a federal law shall regulate,**	
Composition of all parts (Step 7):	These details,	which **a federal law shall regulate,**	require consent of the Bundestag.

Table 1: Example how the sentence is split into preclause, passive construction, and postclause

Additionally, Table 2 shows the described procedure with sentences having more than one passive construction, and how the output of the important steps looks like:

Original sentence:	It shall be tested and it shall be called the configurable device.	
Identified passive constructions and their subtrees (Step 2):	It shall **be tested** and it shall be called the configurable device.	it **shall be called** the configurable device.
Transformed active constructions (Step 7):	**One shall test** it and it shall be called the configurable device.	**one shall call** it the configurable device.
Composition of the generated sentences by inserting the shorter construction into the longer (Step 8):	**One shall test** it and **one shall call** it the configurable device.	

Table 2: Example of how several passive constructions are processed

3.2. Challenges & Approaches

During the implementation of the above-described algorithm, I faced different kinds of challenges and tried various approaches before I came up with the implemented procedure.

Identifying the important components:

Initially, when every word of the sentence was analysed (Step 4) after each other, it sometimes happened that some words were assigned to more than one variable. For instance, a preposition like "in" could be stored as preposition because it has the dependency *prep* and the exact same word can also be part of the *nsubjpass* like "all rooms *in* the building" or of another subtree. If continuing like that, some words might have appeared more than once in the final sentence. My first approach to solve this issue was to check whether there are duplicates or substrings in other components before the composition of the active construction (Step 6), and if so, remove the shorter component. However, if there were some words more than once in the original clause, e.g. auxiliary verbs or prepositions, then one of them was deleted eventually. Therefore, this approach was no longer an option. As a second approach, the indices of the words and their subtrees which have been already allocated, were stored in an array to not analyse them further and to prevent repetitions. This worked well and thus is still used.

Composing the final sentence:

Another challenge was to compose the final sentence after applying the basic conversion of the predicate, subject, and objects. Although in English active sentences, the basic word order is subject–verb–object, additionally dealing with subclauses, prepositions, adverbs, conjunctions, etc. and placing them correctly was the most challenging part. Firstly, I implemented the same structure for every sentence for assembling the final components. Even though the sentences were grammatically correct in most of the cases, especially regarding longer and more complex sentences, it did not work that well. Therefore, my next idea was to identify the basic structure and the basic order of the upper-level components, like subclauses, adverbial components, etc, of a sentence after Step 2 by using consistency parsing. This method parses the sentence into its constituents like Noun Phrase (NP) and Verb Phrase (VP) and then splits each component again into its other components, e.g. NP in Determinator (DET) and a Noun (N). Then, by defining and applying context-free grammar (CFG) rules, the idea was to analyse the structure of the sentence and based on the CFG rule, which fits the sentence grammar, compose the components after the transformation in the right order. One problem here was to define which level of abstraction is suitable for the identification of the basic sentence structure. It should not be too abstract but also not too detailed to only get the most important components. Additionally, CFG rules have to be defined in very detail and have to be able to deal with all different kinds of sentence structures and grammar that might appear in the English language. According to the related work for short and predefined sentences, it works rather well. However, as far as I understood, for longer, more complex sentences, it would be very difficult and not really manageable in my scope. Therefore, after some (unsuccessful) implementation efforts, I no longer focused on that idea but came up with the following and final approach.

I realised that it is normally enough to only transform the passive construction and leave the rest of the sentence as it is, even for longer sentences. If the passive construction is not in the main clause in Step 3 of my algorithm, the original sentence is split into *preclause*, passive construction and *postclause*. Then, only the passive construction, which often only consists of a shorter (sub)sentence, is transformed. After that, the three parts are composed together again to form the final active sentence. Since adverbial sentences, clauses and insertions with no passive construction are not considered and processed further, the algorithm only focuses on the conversion of the passive parts. This is more graspable and less error-prone.

Dealing with sentences with more than one passive construction:

Having found a rather good solution for sentences with one passive construction, dealing with more than one passive construction in one sentence was still difficult. With the approach just described, it was possible to transform the passive construction of the main clause, but not to transform another passive construction in the same sentence. At first, the idea was to implement something like a recursion so that the sentence is split into its subclauses, and then they are all analysed separately and compiled afterwards. But again, at least from my experience, it was difficult to implement the identification of subclauses in a general manner because even dependency parsing, or constituency parsing was not always clear and unambiguous.

Finally, the approach was to identify all passive constructions in Step 2 and apply the transformations for these by just focusing on one passive construction at a time. After that, the generated sentences were then composed together by putting the shorter subclauses into the longest subclause or into the whole clause by using the position indices of the passive construction identified in Step 3.

Dealing with punctuation, capitalization, negations, and prepositions

Some other but smaller challenges were implementing punctuation, capitalization, negations, and prepositions in the right and most generic way. The first two I covered by leaving as much as possible as in the original clause, i.e. checking for a comma as the next token and assigning it to the variable to which the token before was stored or only changing the capitalization of the first word in the sentence when it was not identified during NER. Negations and other rules regarding the verb transformation were just implemented by if – else rules since, in reality, there exist defined rules regarding how and what is conjugated and composed. Regarding prepositions, conjunctions and particles, it is tracked whether it is at the beginning or to which other component (e.g. verb, subject) it is connected. Then, it is stored in a variable and composed accordingly.

When implementing, applying, and finetuning the algorithm, I always ensured to only implement rules that are generic and not only fit this exact sentence. Precisely, the goal was to implement the algorithm as dynamically and adaptive as possible with the rule-based approach.

4. Evaluation

For the evaluation of my algorithm, I considered both the ability to identify passive constructions in inserted sentences as well as the correctness of the passive-to-active conversion. This evaluation can automatically be executed when a file transformation is done (Scenario B). As mentioned earlier, the file must have a column named “mode” with the mode of the sentence, i.e. either passive or active and a column named “references” with the expected active sentence. To be able to evaluate the identification of passive sentences correctly, I added the column “mode” and 15 active sentences to the golden standard, which were also extracted from the same three regulatory documents.

During the evaluation process, the active construction/subclause (i.e. the transformed passive construction) is compared to the same active construction/subclause of the reference sentence. Comparing only the transformed constructions with the expected ones facilitates the comparison of the semantic similarity since the sentences are shorter. Thus, the function applied can analyse the semantic similarity more precisely and is more sensitive to differences and similarities. To analyse the semantic similarity the Sentence-BERT (SBERT) Score is used. This is a modification of the pre-trained BERT network that uses siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine similarity. It performs well in capturing the semantic similarity between sentences. Another possibility would be to use the BERT.Score or SpaCy´s Similarity method. The former is more used for text generation tasks like translation or summarization. The latter is more simplistic than the BERT methods as it relies on averaging word vectors and may not capture complex sentences. Additionally, when implementing all options SBERT seemed to be closest to the human evaluation regarding the semantic similarity of two sentences. Therefore, SBERT and the threshold > 0.95 were chosen to indicate whether a sentence was transformed correctly. The threshold seemed to be appropriate when analysing which sentence had which similarity score.

Overall, precision and recall were used, which were defined as follows:

True Positive (TP): correctly identified as passive and transformed correctly (SBERT-Score > 0.95)

False Positive (FP): wrongly identified as passive and attempted to transform

True Negative (TN): correctly identified as active and thus not transformed

False Negative (FN): (wrongly identified as active and not transformed) PLUS (correctly identified as passive and transformed wrongly (SBERT-Score <= 0.95)

Applying the described procedure on the file "Goldstandard_updated.csv" the final metrics and scores are the following:

TP:	150
FP:	0
TN:	15
FN:	10
Precision:	1
Recall:	0.9375

Since the comparison of the semantic similarity via a predefined method is not always that meaningful, an accuracy rate has been generated by human evaluation. Precisely, every output sentence was compared to its corresponding reference sentence in the golden standard, and if it was correctly transformed (grammatically and semantically), this sentence got a 1; otherwise a 0. To calculate the accuracy, the number of correctly transformed sentences was divided by the number of all transformed sentences, which led to a final accuracy rate of 93,75%.

# of transformed sentences	# of correctly transformed sentences	Overall Accuracy:
160	150	**93,75%**

A detailed overview of each semantic similarity and the result of the manual evaluation can be viewed in the document "FinalEvaluation.xlsx". Therein, to the best of my knowledge and ability, I commented on my assessment and some potential reasons for the wrong conversions.

Moreover, the algorithm was also tested with the golden standard from Rafi. However, the analysis was not that detailed, and it only serves to gain a better feeling for the performance of the algorithm. Following the same procedure as described the following results were gained:

TP:	114
FP:	0
TN:	15
FN:	51
Precision:	1
Recall:	0.6909

The lower performance of the algorithm can be attributed to different methodologies in creating the golden standard. Since Rafi uses an LLM to approach the passive-to-active transformation, the chosen reference sentences resulted in more complex sentences. In cases where an agent wasn't explicitly stated, it sometimes was inferred from the context. Also, there were some slight modifications in reference sentences compared to the original ones that were challenging for a rule-based algorithm to recognize and apply.

Due to the reasons mentioned above, a human evaluation of the false negatives (FN) revealed that 31 out of 51 sentences were actually correct (details can be viewed in the document “FinalEvaluation”). This finding suggests a lower actual FN number and a higher true positives number (TP), probably improving the recall measure. Despite these challenges, the algorithm's effectiveness on Rafi´s golden standard indicates its adaptability beyond my golden standard and its applicability to a variety of sentences.

5. Limitations

Even though the algorithm has an approach to deal with more than one passive construction, the conversion for these sentences does not work as well as if a sentence has only one passive construction. The main reason for that is that then often, the sentences get more complex and nested, which leads to more difficulties in parsing and handling the components.

Investigating the sentences which were not correctly transformed, it can be observed that a) these sentences were sometimes even for a human rather complex and rare constructions or b) there were some errors or ambiguities in dependency parsing from SpaCy. For instance, the preposition “by” does not necessarily always indicate an agent. However, SpaCy identified it as an agent (row 13 when sorting the table on ascending similarity score).

There are some cases where SBERT-Score is below 0.95, even though it would be grammatically and semantically the same as the reference sentence. For instance, if the word order is too different than expected (rows 6 & 9) or if there arise some parsing ambiguities during evaluation (rows 8 & 11). Additionally, for sentences with more than one passive construction, in the reference sentence it is often expected that there is only one construction transformed. However, the algorithm always tries to transform all passive constructions in a sentence. This might lead to differences between results and references and thus to a lower similarity score, even though even more than expected was correctly transformed (rows 7 & 10). Other limitations are symbolics or punctuation, with which spaCy has sometimes some difficulties in parsing them correctly.

These mentioned deviations and errors are very specific, and in order to prevent the code from being too tailored to the golden standard, they were covered just to a certain extent.

6. Conclusion and Outlook

Although the algorithm has rather good accuracy, precision, and recall, there are several issues which could be improved further or tried out to address the mentioned limitations. For instance, CFG or constituency parsing could be used to analyse the passive constructions in a more sophisticated way. Furthermore, a program or library could be used to check the generated sentence for minor spelling or grammar mistakes and even correct them. I tried to integrate Grammarly or another API, but the ones I found were either no longer maintained (Grammarly) or were only usable with a paid subscription. Furthermore, the algorithm could be extended by a machine learning model, which maybe would be able to perform better in the punctuation and the composition, ensuring that in the end, at least a grammatically correct sentence would be generated. Another possible feature could be to derive the agent of the passive construction from the context even if there is no agent mentioned, instead of always using “one” as the default agent. Also, other libraries or NLP tools, such as Stanford Parser or NLTK, could be experimented with to assess whether the parsing could be even more precise.

Overall, approaching the problem of transforming passive long, regulatory, complex sentences with a rule-based algorithm can be seen as beneficial for pioneering ideas and implementations.

Literature

Agrawal, P., Madaan, V. and Sethi, N. (2016). “A novel approach to paraphrase english sentences using natural language processing”.

Fahad, M. and Beenish, H. (2020). "An Approach towards Implementation of Active and Passive voice using LL(1) Parsing," 2020 International Conference on Computing and Information Technology (ICCIT-1441), Tabuk, Saudi Arabia, 2020, pp. 1-5, doi: 10.1109/ICCIT-144147971.2020.9213802.

Ilukkumbura, O. and Rupasinghe, S. (2021). “Sinhala Active Voice into Passive Voice Converter using Rule Based Approach with Grammar Error Correction”.

Nohimovich, D. and Jin, Z. (2018): “pass2act”. GitHub. Online available: https://github.com/DanManN/pass2act, last checked on 28.01.2024

Pawale, Dr. D. A., Pokale, A. N., Sakore, S. D., Sutar, S. S. and Kshirsagar, J. P. (2015). “Sentence correction for english language using grammar rules and syntax parsing.”

Tayal, M. A., Raghuwanshi, M. M., and Malik, L. (2014). "Syntax Parsing: Implementation Using Grammar-Rules for English Language," 2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies, Nagpur, India, 2014, pp. 376-381, doi: 10.1109/ICESC.2014.71.