Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

ActiveAnno: Flexible and Efficient Document Annotation Tool with Active Learning Integration

Masterthesis
at Research Group Language Technology, LT
Prof. Dr. Chris Biemann

Department of Informatics
MIN-Faculty
Universität Hamburg

submitted by
Max Wiechmann
Course of study: Informatik
Matrikelnr.: 7089736
on
27 July 2020

Examiners: Prof. Dr. Chris Biemann
Dr. Seid Muhie Yimam

Abstract

This thesis presents the novel document annotation tool ActiveAnno. The focus of ActiveAnno is to be a well-functioning general-purpose annotation tool for document-level annotations. Through high configurability, users can optimize for high-quality annotations and efficiency. One of the main features to enable this is a machine learning integration with the ability to provide pre-annotations and to integrate ActiveAnno into an active learning process. In this context, two experiments with ActiveAnno were conducted in an industry setting. In the first experiment, the effects of document-level pre-annotations on annotation quality and efficiency were analyzed in comparison to the manual annotation process. The results show a 28% decrease in annotation duration using pre-annotations. At the same time, annotator accuracy and inter-annotator agreement show no significant changes. The second experiment compared model performance for incremental learning with random sampling and active learning with uncertainty sampling for a new annotation task in ActiveAnno, using its machine learning integration capabilities and a fastText-based external classification service. The results show a worse performance for a model trained with active learning when not providing any pre-trained word vectors to it. Using pre-trained word vectors, both, models trained with random sampling and uncertainty sampling perform better overall, without a significant difference between each other. These results show that the choice of classifier is relevant in an active learning setting and that using unsupervised methods in form of pre-trained word vectors can prevent the sampling bias of uncertainty sampling in this context.

Zusammenfassung

Diese Masterarbeit stellt ein neues Annotationstool für Dokumente namens ActiveAnno vor. Der Fokus von ActiveAnno liegt darauf, ein gut funktionierendes, vielseitig einsetzbares Annotationstool für Annotationen auf Dokumentenebene zu sein. Durch hohe Konfigurierbarkeit können die Nutzer sowohl für hochqualitative Annotationen als auch für einen effizienten Annotationsprozess optimieren. Eine der wichtigsten Funktionen, um dies zu ermöglichen, ist eine Machine Learning Integration mit der Möglichkeit, Pre-Annotationen bereitzustellen sowie ActiveAnno in einen Active Learning Prozess zu integrieren. In diesem Kontext wurden zwei Experimente mit ActiveAnno innerhalb eines Industrieumfelds durchgeführt. Im ersten Experiment wurde der Effekt von Pre-Annotationen auf Dokumentenebene mit Bezug auf Qualität der Annotationen und Effizienz analysiert und mit dem manuellen Annotationsvorgang verglichen. Die Ergebnisse zeigen eine 28% kürzere Annotationsdauer mit Pre-Annotationen. Gleichzeitig gab es keine signifikanten Änderungen bei der Genauigkeit der Annotatoren oder dem Inter-Annotator Agreement. Das zweite Experiment vergleicht die Performance von Models für Incremental Learning mit Random Sampling und Active Learning mit Uncertainty Sampling für eine neue Annotationsaufgabe in ActiveAnno, unter Nutzung der Möglichkeiten zur Machine Learning Integration und einem auf fastText basierenden, externen Classification Service. Die Ergebnisse zeigen eine schlechtere Performance für ein mit Active Learning trainiertes Model, wenn keine vortrainierten Wort-Vektoren bereitgestellt wurden. Unter Verwendung von vortrainierten Wort-Vektoren performen sowohl Models trainiert mit Random Sampling als auch Uncertainty Sampling besser, ohne, dass es zwischen beiden signifikante Unterschiede gibt. Diese Ergebnisse zeigen, dass die Wahl des Classifiers im Kontext von Active Learning relevant ist und Unsupervised Learning Methoden in der Form von vortrainierten Wort-Vektoren den Sampling Bias von Uncertainty Sampling in diesem Fall verhindern können.

Acknowledgement

This thesis was done in cooperation with the Qualitize GmbH. I want to thank them for their support, for the opportunity to create this open-source tool, and for the ability to use this tool to conduct research on annotation tools in an industry setting.

Contents

Chapter 1

Introduction

Annotation tools are software programs that provide users, the annotators, a user interface to annotate some documents with additional information. Documents can be texts, images, videos, audio files or have a complex multimodal structure. Each type of document has specific kinds of annotation targets: Texts can have annotations for specific substrings or spans. Images can have annotations for a subset of pixels, audio files for a specific time range and videos can be annotated for time ranges, frames or subsets of pixels in those frames. On a higher abstraction level, all documents, regardless of their type, can have annotations on the document as a whole, called the document level. This thesis focuses on **document-level annotations** for **text documents**.

Typical use cases for annotation tools are found in research as well as in an industry context. In natural language processing research, annotation tools can be used for tasks such as named entity recognition (NER) [7]. In social sciences, content analysis [14] can be used to assign codes to documents, for example to extract important topics from interview transcripts in order to generate quantitative data. In supervised machine learning or classification tasks, which are used in research and industry alike, a machine learning model learns to predict the correct label for a given input, like a text document. An example for this would be a spam filter in a mail program, assigning a *spam* or *no spam* label to each new email based on previously labeled emails. Both codes and labels can be seen as analogous to annotations. In all those cases, there is a human involved at some point to annotate the documents, which would typically be done through an annotation tool.

While annotation tools can be used to train a machine learning classifier, these classifiers can also be integrated into annotation tools to create a more efficient annotation process. If a well-trained machine learning model already exists, it can be leveraged to *pre-annotate* documents and display those in the user interface. Alternatively, the model can be used to automatically annotate all or a subset of the documents. When no trained machine learning model exists at the beginning of an annotation task, **incremental learning** can be used to iteratively train new versions of machine learning models based on the annotations created by a human in the annotation tool. A special version of incremental learning is **active learning**, where a deliberate choice is made about which documents to annotate first in order to make the machine learning model learn faster with less training

samples required. Finally, the newly trained model can be used to automatically annotate either all or a subset of the documents, greatly reducing the manual labor required to finish the annotation task.

1.1 Context

During the Web Interfaces for Language Processing Systems project at the University of Hamburg, the author created the document annotation tool ActiveAnno[1], which was done in cooperation with a local company. ActiveAnno is an open-source annotation tool with focus on document-level annotations. For this master thesis, ActiveAnno was improved and extended to have a machine learning integration and active learning capabilities to pursue the goal of creating high-quality annotations while minimizing effort. The thesis was also done in cooperation with the local company, offering the ability to conduct experiments with annotators, tasks and data from an industry environment.

1.2 Research Questions

By leveraging the ability to conduct experiments in a practical setting, two research questions are investigated. Both revolve around using machine learning to improve annotation processes with regards to annotation quality and efficiency.

Research Question 1 What is the effect of pre-annotations in the form of generated document-level annotations from a machine learning model on the annotation quality and the efficiency with which these annotations are created?

Research Question 2 How does active learning with uncertainty sampling compare to incremental learning with random sampling with regards to the generated annotation quality given a new machine learning model without preexisting training data?

1.3 Contribution

The contribution of this thesis is fourfold.

First, ActiveAnno is extended by a machine learning integration and active learning capabilities, further providing users of ActiveAnno with the ability to design efficient annotation processes.

Secondly, ActiveAnno is used in an industry setting as well as for the research purposes of this thesis and improved for requirements from both contexts, furthering the quality and stability of ActiveAnno as an open-source document annotation tool.

[1] `https://github.com/maxmello/activeanno`

Thirdly, two experiments are conducted, answering the research questions above about ways to efficiency create high-quality annotations for a practical use case.

Lastly, through the experiments, it is also demonstrated how to design, conduct and analyze controlled experiments with ActiveAnno as a reference for other researchers.

1.4 Terminology

The specific terminology used for concepts like documents and annotations can vary between papers or research contexts. Therefore, the terminology used in ActiveAnno and this thesis is defined here.

A **document** is a single record of related data, which are of basic data types such as strings, numbers and Boolean values. A document contains exactly one **document text**, which is the main attribute of the document and of type string. All other data attributes are regarded as **meta data** of the document, for example an associated creation time or the name of the creator. Every document can be annotated for any number of annotation **projects**. A project is created for a specific annotation task and defines among other properties an **annotation schema**. The annotation schema is an ordered list of **annotation definitions** associated with their project-specific configuration. The annotation definition specifies what has to be annotated about a document, for example the sentiment of a comment as a choice of predefined **tag set options**, the key topics of a news item defined as an extensible list of tags, or the usefulness of an app review expressed through a number between one and ten. The **annotation** is the combination of the annotation definition and the created **annotation value**, for example the value *positive* for the annotation definition *sentiment*. The set of all annotations produced for an annotation schema is called an **annotation result**. Each document can have a completely different set of annotation results for each project associated with the document.

In the context of machine learning, the term **label** is seen as analogous to the annotation. A document used as training data might also be called a (training) **sample** or instance. For text classification, the term **class** is seen as analogous to a tag set option as defined in the context of annotation tools.

1.5 Thesis Structure

The remaining structure of this thesis is as follows: The second chapter describes the related work regarding annotation tools with support for document-level annotations, text classification, active learning and the effects of using pre-annotations. The third chapter gives an overview of the concepts, architecture and functionality of ActiveAnno. The fourth chapter presents the two experiments conducted using ActiveAnno in an industry setting and the fifth and final chapter summarizes the findings and gives an outlook on the future work regarding ActiveAnno.

Chapter 2

Related Work

To provide context for the design decisions of ActiveAnno, this chapter gives an overview of existing annotation tools with relevance for document-level annotations. Additionally, relevant topics from machine learning, specifically text classification and active learning, are presented for the understanding of the machine learning integration capabilities of ActiveAnno as well as the experiments. Finally, for the intersection of machine learning and annotation tools, previous work on the effects of pre-annotations is described.

2.1 Annotation Tools

Neves and Ševa [15] conducted an extensive review of document annotation tools in 2019, collecting 78 tools in total and comparing 15 which met their minimum criteria. Their five basic criteria for considering a tool were the availability of the tool, it being a web application, it being installable in under two hours, it working without errors while using it, and it allowing for the configuration of a schema. The 15 selected tools were then evaluated on 26 criteria from four categories: Publication criteria, technical criteria, data criteria and functional criteria. The technical criteria included the date of the last version, the availability of the source code, the ease of installation, the quality of the documentation, the type of licence and whether or not the tool is free of charge. The data criteria were comprised of the format of the schema (e.g. configured through a GUI or configuration file), the format for importing documents and the format for exporting annotations, e.g. JSON or XML. Some functional criteria not related to span-level annotations included the ability to support multiple users or teams, analyzing inter-annotator agreement, the ability to display pre-annotations in the context of active learning and the support for document-level annotations.

Their original goal for the paper was to evaluate tools capable of annotating on a document level, but they expanded their review to those tools that only support span-level annotations, because only five of the selected tools supported document-level annotations. Those tools are MAT, MyMiner, tagtog, prodigy and LightTag.

MAT [1] is designed for what they call the “tag-a-little, learn-a-little (TALLAL)

loop" to incrementally build up an annotation corpus through manually annotating some documents, training a new machine learning model, automatically annotating the documents, correcting those automatic annotations and repeating the process. It has an extensive documentation, but was not intuitive to use according to Neves and Ševa. According to the online documentation, it is a research prototype and not ready to be used in production. In the evaluation of Neves and Ševa, MAT scored a 0.6 of a maximum of 1, being slightly below the average of 0.62.

MyMiner [17] is an online-only tool without a login or user system. Its main purpose is to classify scientific documents in a biomedical context. The document-level annotation process is based on uploading text files, defining the possible labels and annotating every file with one of the possible labels. The results can be downloaded as a text file containing the chosen label and the uploaded file content. The configuration aspect is limited to defining the possible label strings. As the tool is intended for scientific documents, the user interface is designed to display a paper title and content. Neves and Ševa scored MyMiner with a 0.52.

The tool tagtog [2] is a commercial annotation tool with the free plan allowing online use only. Using the free plan, private datasets and the machine learning functionality are not available. The online documentation[1] shows a feature-rich application. Projects can be created with annotation guidelines, document labels, webhooks and project members. Metrics include inter-annotator agreement and document-level annotation distribution statistics. It also supports an API secured through HTTP Basic authentication. The paid version of tagtog supports annotation automation through machine learning models that will be trained from manual annotations automatically, if enabled. It scored a 0.6 in the evaluation of Neves and Ševa.

The fourth tool supporting document-level annotations is prodigy[2], which is fully commercial and does not have a free version. The functionality can be evaluated through the available live demo and documentation[3]. The setup of the tool as well as the configuration of an annotation schema is done through a command line interface and the knowledge of Python is assumed to do the setup[4]. It supports different types of documents, including texts, images, audio and video. The annotation user interface shows the progress and history of annotated documents, and every document has up to four available basic actions in addition to the specifically defined annotation: Accept the annotation, reject the annotation, ignore the document and go back to the previous document. Which actions are available to the annotator is configurable. Support for active learning exists and the application is extensible through custom Python code, allowing for customized machine learning. Neves and Ševa report a positive user experience, though the tool scored a below average 0.56 in their evaluation.

LightTag[5] is the last evaluated tool which supports document-level annota-

[1] `https://docs.tagtog.net`
[2] `https://prodi.gy/`
[3] `https://prodi.gy/docs`
[4] `https://prodi.gy/docs/faq`
[5] `https://www.lighttag.io/`

tions. It is also commercial and lists a number of useful features on their website[6]: Document classification and tagging, which are part of the free plan, and more advanced features like teams, multiple annotators per document, statistics about annotator performance, user roles and inter-annotator agreement analysis, which are part of the paid plans.

The best-performing tool in the comparison of [15] is WebAnno [25] with a score of 0.81. While is does not support document-level annotations directly, there exists a workaround using zero width annotations[7].

The features of WebAnno include user management and a role system with administrators, project managers, annotators and curators [4], where certain user interfaces are only accessible to for certain roles. Project managers can define projects and monitor progress and inter-annotator agreement. Annotators have access to the annotation interfaces, where pre-annotated and un-annotated documents can be annotated. Curators can choose the correct annotation between the alternatives created by annotators, or freely annotate the document themselves, giving them authority over the final annotations. Regarding the project setup, project managers can define an arbitrary number of annotation layers, for example for part-of-speech tagging or named entity recognition. Created annotations are immediately persisted to prevent any unintended loss of progress. It also provides an inbuilt generic machine learning component [26] for automatic annotation suggestions for the annotation layers *lemma, NER, POS and co-ref*[8]. It does not require the connection of an external machine learning component to function. WebAnno can be deployed in a variety of ways, including Docker.

Combined with the fact that WebAnno is still under active development, that it is open-source and that it has a great online documentation[9], makes it a useful, feature-rich general-purpose annotation tool. While it does not directly support document-level annotations, most of the features not directly related to span annotations are still very important to consider for the development of any annotation tool.

Another annotation tool with limited support for document-level annotations is Doccano [13], though it is not mentioned in the evaluation of Neves and Ševa. The open-source tool currently supports three distinct annotation tasks: text classification, sequence labelling and sequence to sequence tasks. The text classification task allows users to annotate text documents with pre-defined classes, while the sequence to sequence task allows to create any text as a document-level annotation. There is no support for other types of annotations such as numbers or user-extensible tags. There is no support for automation, though the REST API could be used to integrate the tool in an automation process. Doccano can also be deployed with Docker.

[6] `https://www.lighttag.io/features`

[7] `https://github.com/webanno/webanno/issues/923`

[8] `https://webanno.github.io/webanno/releases/3.6.4/docs/user-guide.html#sect_automation`

[9] `https://webanno.github.io/webanno/`

2.1.1 Discussion

As seen from the described tools, most of the commercial tools are usually general-purpose and feature rich, because of the commercial incentive to sell as many licenses as possible. But in a research context, having open-source applications free of charge is valuable, especially if the researchers want to promote an open source and open science mindset. On the other hand, most non-commercial tools have less features and are often limited in scope, as they are developed for a specific research purpose. In general, the number of tools with focus on span-level annotations greatly outnumbers the tools with first-class support for document-level annotations. WebAnno is the notable example of a tool which is open-source, widely used, feature rich and has at least a workaround for document-level annotations. Still, Neves and Ševa write: "Therefore, there is much room for improvement of tools with such features, or even the development of a tool specifically for this purpose." (Neves and Ševa, 2019, p. 13) with regards to document-level annotations. As most tools are developed either for a commercial industry setting or for a more narrow research setting, having a collaboration of industry and research as with ActiveAnno can be very valuable to get a feature-rich general-purpose annotation tool which exists in an open-source setting and in a research context.

So instead of starting with span annotations and then creating document-level annotations as an afterthought, ActiveAnno is intended to be document-level first. Meanwhile, it is still designed to be extensible for span-level annotations and hybrid annotations later. All the criteria and features which make a good annotation tool, like the criteria defined in [15] or the notable features of the mentioned tools, but are not specific to span-level annotations, can still be used as inspiration to create a useful general-purpose document-level annotation tool.

2.2 Text Classification

Mirończuk and Protasiewicz [12] provide an extensive review over the field of text classification. In essence, text classification is the automatic assignment of predefined labels (also called classes) to a text.

Most commonly, this is done in a *supervised* way based on already labeled training samples. As this work focuses on text documents in particular, text classification is what will be used to automate parts of the annotation process.

While Mirończuk and Protasiewicz list an array of training methods like decision trees, support vector machines (SVM), or neural network classifiers, the implementation details of such classification methods are not the focus of this work. To anticipate Section 3.5, ActiveAnno supports machine learning integration through an HTTP API where any classification method can be integrated. Still, for the experiments described in Chapter 4, the choice for a text classification approach is required. As such a choice should be able to be arbitrary in ActiveAnno, so was the choice for the method used for the experiments: fastText[9], a library that supports text classification through the command line as well as a Python library. Is uses word embedding techniques which generally performs better for Active Learning

(Section 2.3) tasks as compared to simpler methods like TF-IDF [11].

In [21], the authors Sokolova and Lapalme describe four types of classification tasks: Binary, where the choice is between two classes; multi-class, where the choice is between one of many non-overlapping classes; multi-labelled, where the choice is between several of many non-overlapping classes; and hierarchical classification, where one class is chosen from many classes which are structured in a hierarchical manner. They then analyze the various type of performance measures for these tasks. A basis for most measures are the concepts of *true positives*, *false positives*, *true negatives* and *false negatives*. Defined on a binary classification task with a positive and negative case, true positives are those samples correctly classified as positive, false positives those falsely classified as positive, true negatives those correctly classified as negative and false negatives those falsely classified as negative. *Precision* is calculated by dividing the *true positive* cases by the sum all positive classifications, expressing the proportion of positive classifications which are correct. In contrast, *Recall* is calculated by dividing the *true positives* by all actually positive samples, expressing the ability of classifiers to identify positive classes. Precision and Recall are often combined through their harmonic mean as the *F-score* to use as a general indicator of a classifiers performance. Another measure is *accuracy*, which describes the effectiveness of a classifier, expressed through the proportion of correct classifications over all classifications. These measures are defined on binary classification tasks, but can be transferred to the multi-class task by building averages over the multiple classes. For multi-labelled classification tasks, some measures include the *Exact Match Ratio*, which is the proportion of exact classifications for all classes on a document; and the *Labelling F-Score*, which also considers partial matches.

To evaluate a classifiers predictions, a source of what is actually the correct classification is required. This is called the gold standard. As described in [8], often there is no perfect gold standard. Instead, the performance of a classifier is compared to classifications done by humans. To increase the confidence in the gold standards quality or reliability, often multiple humans (annotators in the context of annotation tools) are used to create the gold standard by using their majority opinion. The degree to which multiple annotators agree in their annotations (or classifications) is called *inter-annotator agreement* (IAA). One way to measure it is by using *simple agreement*, meaning the proportion of documents where the annotators agree [8].

2.3 Active Learning

Active learning is a strategy for training a machine learning model where the model is integrated into the choice of which training samples to label next [3]. This is useful if the amount of potential training data is large and there does not yet exist any of it. By actively selecting which documents to annotate first, the goal is to train the model using less manually labelled samples overall as compared to selecting samples at random. This is based on the assumption that some documents

are more informative for the model than others. For example, a text classifier for sentiment which already has seen the text "Awesome!" multiple times, every time labelled with the value *positive*, will gain less information from that exact same text again as compared to a new text like "This was very nice." which it has never seen before.

One difference in active learning approaches is the *query strategy*, the way to choose which documents to sample next for labelling. In [18], different query strategies are described. One approach is to use *uncertainty sampling*, which queries those documents first where the model is most uncertain about the appropriate label. For models that support assigning a probability to a prediction, this probability can be used as the uncertainty value. For example, a binary classification task with two labels might return a probability of 0.7 for *label A* and 0.3 for *label B* for one document, and 0.45 for *label A* and 0.55 for *label B* for another document. In this case, the model is more uncertain about the second document, because the highest probability value for the second document (0.55) is smaller than for the first document (0.7). Another query strategy is called *query-by-committee* [20], where multiple models predict labels and the documents with the highest disagreement among the committee of models are queried first. Zhu et al. [27] use active learning in conjunction with semi-supervised learning, where the initial documents for a semi-supervised approach are sampled with active learning. The addition of semi-supervised learning includes the unlabeled data into the decision about which documents to label next, which can prevent sampling only the outliers, as discussed in [18].

In addition to the query strategy, another decision to make is how often to query the model for which documents to label next. In the best case scenario, this can be done after every labelled document. In practice, the model might take too much time to update after every new training sample, which makes querying and labelling in batches of documents an alternative to ensure a faster labelling process. As described by Settles, this might not work well because of overlap between the documents inside a batch, which reduces the actual information gain per labelled document [19].

That active learning is actually effective was for example demonstrated by Tong and Koller [23], who showed how actively selecting which texts to label next for training data reduces the need for the total number of training instances while using support vector machines for classification. Though Varghese et al. [24] show through simulations that while active learning can decrease the training effort, it can also introduce sampling bias which can result in worse model performance after all, reducing the benefits of using active learning.

2.4 Effects of Pre-Annotations

When combining machine learning with annotation tools, one application is to use the automatically generated annotations as pre-annotations, meaning preselecting the inputs of the annotation interface from the machine learning generated anno-

tations. Therefore, instead of coming up with the annotations themselves, the annotators are only required to correct the pre-annotations if they are incorrect. Else, they can just accept the pre-annotations as correct, which decreases the number of required interactions with the user interface for the annotation process. Multiple studies analyzed the effect of span-level pre-annotations on the annotation process and annotation results.

Rehbein et al. [16] used pre-annotation in the context of word sense disambiguation. They compared the F-Score for annotators between a condition with pre-annotation and one without. Overall, the annotators achieved a statistically significant better F1-Score with pre-annotations, even when those were noisy. Though they did not find a statistically significant impact on annotation time. Because they did not control for the order of conditions, they did not prevent the learning effect from having an impact on the results. F1-Scores on the pre-annotated condition were lower for those subjects which received that condition first. This learning effect could also have impacted the missing reduction in annotation time.

Another study was done on the effect of pre-annotations for part-of-speech tagging [5]. They showed an increase of accuracy and inter-annotator agreement when using pre-annotations. They also mention that annotators do show biases in annotating, which they should be notified about to improve annotation quality. Even a POS tagger with a small number of training data was able to significantly decrease the annotation time.

In the context of annotating named entities in clinical data, Lingren et al. [10] compared results for dictionary-based pre-annotations with no pre-annotations and found time saving ranging from 13.85% to 21.5%. They did not find any statistically significant difference for inter-annotator agreement or accuracy. Also in the medical domain, South et al. [22] evaluated the effect of pre-annotations with regard to a de-identification task on clinical texts. Compared to the previous examples, they found neither an improvement in annotation quality nor annotation speed compared to fully manual annotation.

Finally, Gobbel et al. [6] combined pre-annotations with an active learning approach for speeding up the annotation of clinical texts. They used batches of 19 to 21 documents for the active learning task. The first batch had no pre-annotations. Every subsequent batch used pre-annotations based on the previous batches of generated training data. Their tool, RapTAT, achieved up to 50% total time savings for the annotation task, with increased speed for each iteration of the active learning process, which did not occur with the manual annotator control group. They also report significantly higher inter-annotator agreement and did not report any bias in the resulting annotations.

2.4.1 Discussion

All the mentioned work on the effect of pre-annotations were done for various span-level annotation tasks. In general, pre-annotations appear to be able to decrease annotation time as well as improve accuracy and inter-annotator agreement, even though not every study came to the same conclusions. For example, the task in

[10] might have been to complex for pre-annotation to have any effect. Therefore, it is of interest how pre-annotations influence annotation speed and quality for document-level annotations, which will be examined in Section 4.3.

Chapter 3

Tool Design

The criteria defined by Neves and Ševa as described in Section 2.1 give a baseline for design decisions regarding annotation tools: They should be open-source, easy to install, configurable and support standard formats. Furthermore, features like user or role management, statistics such as inter-annotator agreement and some form of machine learning integration can be important for some use cases. As a general-purpose annotation tool, the aim should be to support as much use cases as possible in a reasonable scope. As the focus of ActiveAnno is currently only on document-level annotations, this changes the requirements compared to tools supporting span-level annotations in important aspects. For example, texts that require only document-level annotations are probably shorter, compared to tasks on longer texts such as POS tagging. Texts that could require document-level annotations are single sentences, tweets or internet comments. User generated content like tweets might also be produced regularly, so annotation projects might be based on a stream of data instead of a one-time upload of documents more often. To support this, a good API becomes more important. At the same time, features like the ability to curate annotations (WebAnno), deployment via Docker (e.g. WebAnno, Doccano) and the continuous retraining of machine learning models (MAT) are equally useful for document-level annotation tasks as they are for spans. To guide the design of ActiveAnno, the specific features were abstracted into five design goals.

Annotation Quality and Efficiency To be a general-purpose tool that can be used in place of specialized annotation tools, which often would have to be programmed first, a main requirements is the provide control mechanisms to ensure high-quality annotations and an efficient annotation process. Depending on the use case, one might be more important than the other. For example, when working with a limited amount of documents that need to be annotated thoroughly, the responsible person might want to set up an annotation process where annotations need to be built from the agreement of multiple annotators. In another case, where the goal is to train a classifier with an accuracy of 85% on a very large dataset, it might be enough to use one well-trained annotator and make use of an active learning process to efficiently train that classifier. Users of ActiveAnno should be

able to tune their annotation project in a way to optimize for both aspects as required by their specific situation.

Flexibility and Configurability An annotation tool should be flexible and configurable to be useful in a wide range of application scenarios. Often, researchers will program their own prototype of an annotation tool for their task. Instead, it should be easier to deploy and configure ActiveAnno, without the need to modify the code of ActiveAnno in any way.

Functionality and Usability For ActiveAnno to be worthwhile to use, it needs to have production quality to elevate itself from specialized prototypes. It should provide a high baseline for an efficient annotation process that makes creating high-quality annotations easy, for example through a good user experience with a modern, responsive user interface. Annotators should be prevented from creating accidental errors, for example through validating inputs based on defined limitations on allowed values. Annotations made should not be lost when refreshing the page or closing the browser.

Extensibility and Open Source As new use cases for ActiveAnno arise, it should be as easy as possible to extend the software to allow for new use cases to be supported without complicating the software for existing users and without the code itself becoming worse over time. This is done through use of the right architecture and software design patterns, but also through an open-source development approach, allowing for contributions and a transparent development process.

Interoperability and Installability Based on the principle of low coupling, high cohesion, ActiveAnno should do one thing: Be a tool for annotating documents. Other functionality like specific machine learning implementations or authentication providers should be added by integrating ActiveAnno through interfaces, namely REST APIs and webhooks. This allows ActiveAnno to be integrated into existing software stacks or to build independent services around ActiveAnno. To support this, ActiveAnno should be easy to install or deploy.

The remaining chapter documents the decisions made on design, architecture, implementation and features to give an understanding about the concepts and capabilities of ActiveAnno.

3.1 Software Architecture and Technologies

ActiveAnno uses a client-server architecture. The client is a web application written in Javascript using the user interface library React[1]. It makes use of modern

[1] `https://reactjs.org/`

and popular libraries like redux[2] for application state management, flow[3] for type checking in Javascript and uses Material UI components[4] based on the Material design system[5] for a familiar and modern design.

The backend is built with the lightweight Ktor[6] framework, using the Kotlin programming language[7] running on the JVM. Client and server communicate over HTTP through a JSON API. Additionally, the backend provides a JSON API for external services to communicate with, e.g. for importing documents. MongoDB[8] is used as the data storage, e.g. for documents, annotations and project configurations. The backend is written as a stateless service, meaning there is no server-side session management or global state inside the application used for communicating with clients. This way, multiple instances of the same service could be deployed into a service cluster. For this, ActiveAnno uses JSON web token (JWT)[9] as a stateless authentication mechanism.

3.2 Web Application

The ActiveAnno web application is a responsive, modern single page application. It is fully internationalized, currently supporting the languages English and German. By using a persistent web database, the application state and therefore the created annotations are automatically persisted. Authentication can be disabled when running locally, while still supporting a multi-user login. In that case, anybody can log in with any name and will be treated as a super user. When authentication is activated, an external service needs to be provided for JWT verification. A configurable role system allows for the proper authorization of endpoints: The two main roles are user and manager. A user can be annotator or curator, a manager has the ability to edit projects as well as analyze the results. There are also the roles producer and consumer to allow the protection of the API of ActiveAnno. Producers are allowed to push data into ActiveAnno while consumers are allowed to read the annotation results through the export API. Lastly, the admin role allows access to internal APIs and to an admin panel for user management and application configuration, which is planned for the future. Though slightly modified, the user roles were inspired by WebAnnos user management[10].

The application can be deployed using Docker and can be configured through the use of environment variables to allow to connect the database, configure the JWT security and set the port and host. For the frontend, the theme colors are also

[2] https://redux.js.org/
[3] https://flow.org/
[4] https://material-ui.com/
[5] https://material.io/
[6] https://ktor.io/servers/index.html
[7] https://kotlinlang.org/
[8] https://www.mongodb.com/
[9] https://jwt.io/
[10] https://webanno.github.io/webanno/releases/3.6.5/docs/user-guide.html#sect_users

configurable through environment variables, allowing them to be adapted to existing designs. By default, ActiveAnno generates an example project, which was also used for any screenshots in this chapter. This can be disabled through environment variables as well.

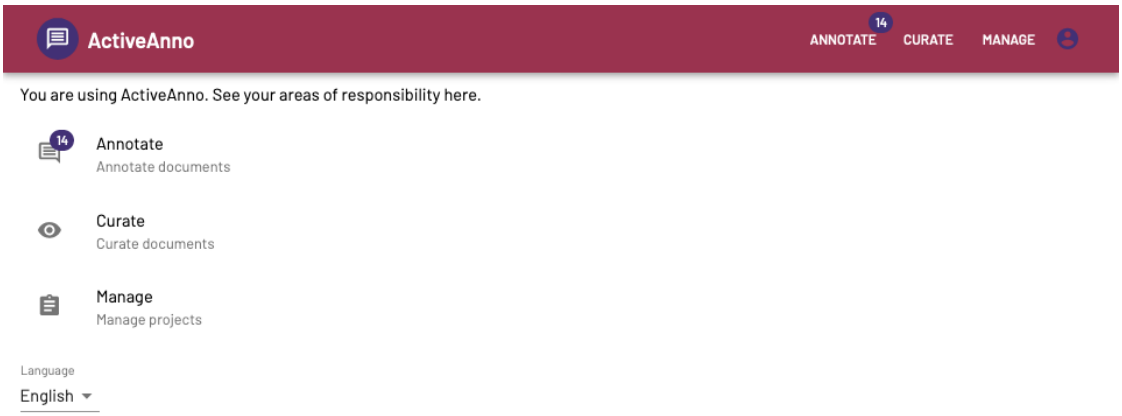


Figure 3.1: The home page of ActiveAnno for a user with complete access to all subareas.

Figure 3.1 displays the home page of ActiveAnno with the three main subareas: Annotate, curate and manage. The manage area allows users with the manager role to edit annotation definitions, projects, upload documents and analyze annotation results for projects.

Figure 3.2 shows the UI for editing the basic properties of an example project, like name, description and the users with access to the project. In addition to these basic properties, it is also required to configure the filter condition to query documents from the database, defining which documents are relevant for the project, as well as the field and order of how to sort documents when querying the database. Both the filter and sort inputs translate to MongoDB queries internally. Additionally, the manager of the project can configure the annotation schema (see Subsection 3.4.1). From this and the *document mapping* step, where the manager defines which part (meaning JSON keys) of the imported documents are relevant for the project, the layout for the annotation task gets generated. It is future work to make the layout fully customizable as well. The default generated layout shows all the metadata, the document text and all the annotation definitions with their default input type. Figure 3.3 shows an example layout for the annotation view. The layout could also be customized by directly calling the JSON API used by the frontend, if necessary. Lastly, the manager can configure how annotation results are able to be exported: Either through the REST API, webhooks, or both. See Subsection 3.3.2 for more details on the API.

After a manager created a project and documents were imported into ActiveAnno, the annotators can then annotate the documents according to the project setup in the annotate subarea, as shown in Figure 3.3. Depending on how the project is set up, the annotated documents might be checked and possibly overwritten by a curator in the curate subarea. An example screenshot for the curate

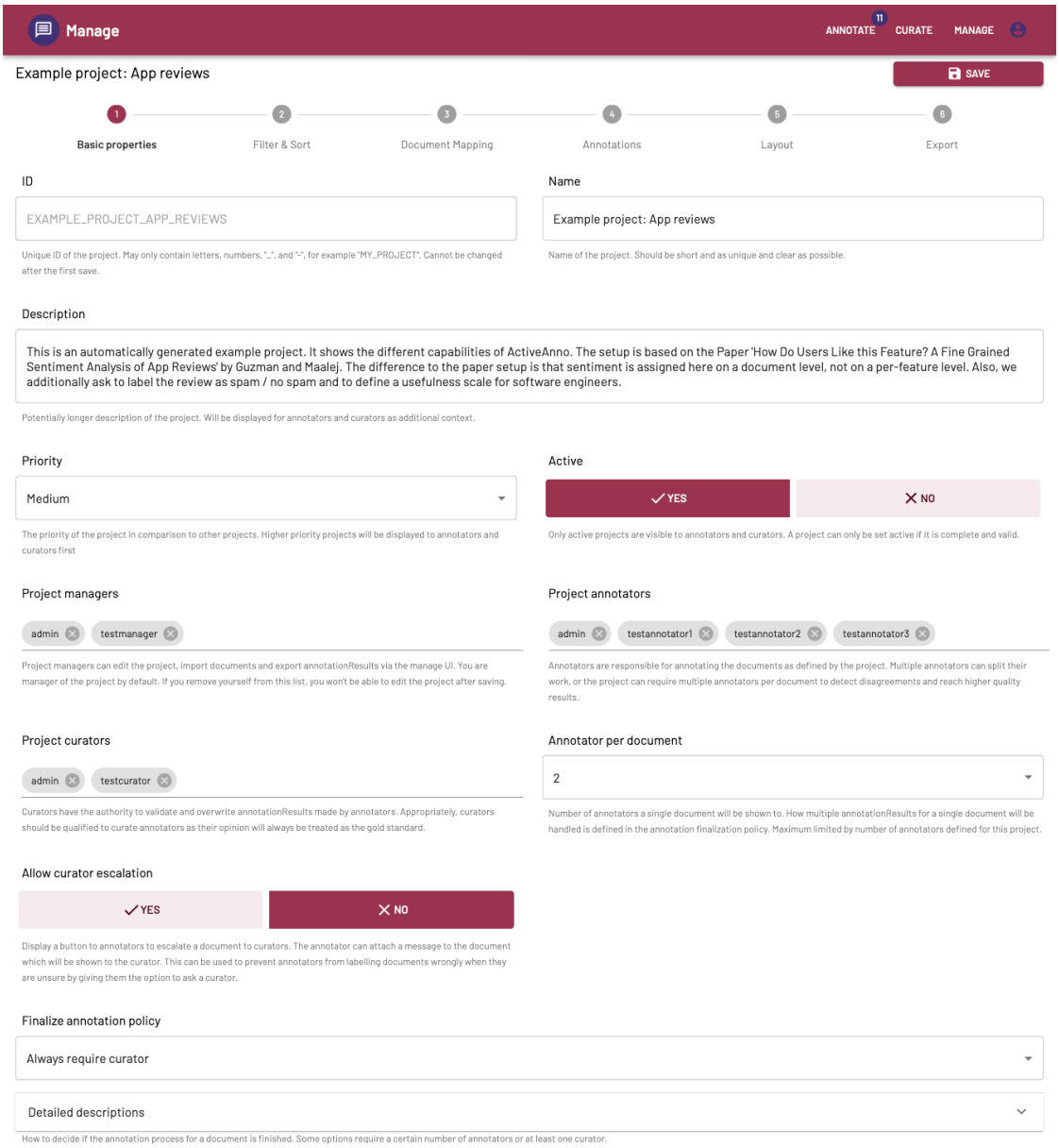


Figure 3.2: A screenshot of the edit project UI in the manage subarea of ActiveAnno. Specifically, this screenshot shows the *Basic properties* step of editing a project.

subarea is shown in Figure 3.4. In addition to the normal annotate UI, curators can see all previously created annotation results. Curators have the authority to decide if an annotation created by an annotator is correct, and annotations created by curators are therefore also treated as correct. This means, ActiveAnno supports a two-step annotation process: First the documents get annotated and then either automatically merged into a final annotation result through an agreement logic, or they will be checked and finalized through a curator. All this is configurable on a per-project basis and is described further in Subsection 3.3.1.

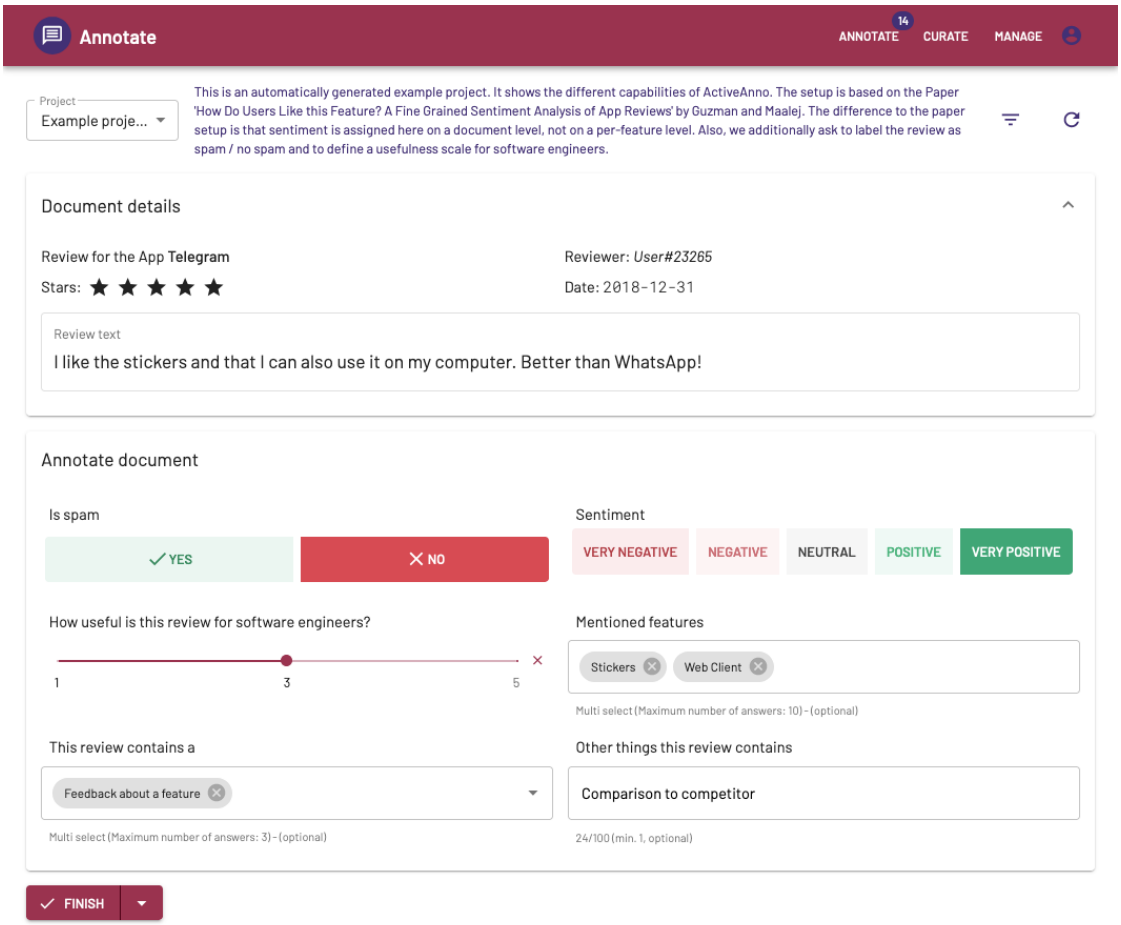


Figure 3.3: A screenshot of the annotate UI for an example project.

After annotation results were created, they can be analyzed by managers in the analyze project UI. Figure 3.5 show an example chart of the accuracy statistics for an example project. In addition to accuracy, the inter-annotator agreement (simple agreement with exact matching), as well as the annotation duration is available as charts in the UI. There is also a table with each individual document, showing the correctness of every annotator and if the annotators agree for the document. The UI has filter inputs to restrict analyzed annotation results by annotator, date range as well as more fine grained filters.

3.3 Documents and Annotations

Figure 3.6 shows an UML diagram of the data models central to ActiveAnno as well as their relationships. The document (`Document` in Figure 3.6) is the model holding the imported document data. A document is independent from any project or other documents. Every document can be associated with any project through the data stored inside the `ProjectAnnotationData` model, which maps annotations for a project to the document. All annotations from a single source are

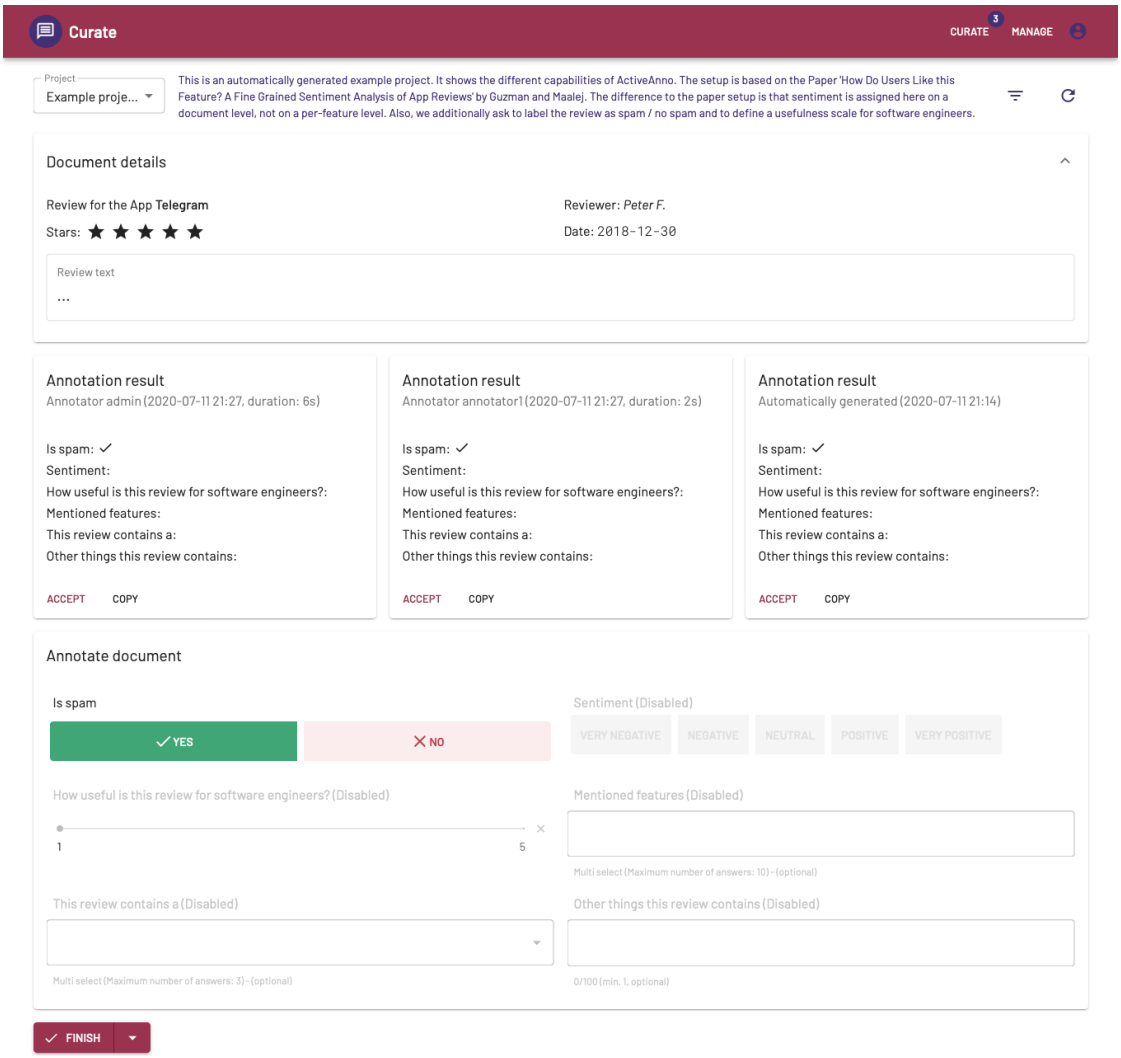


Figure 3.4: A screenshot of the curate UI for an example project.

grouped together as an annotation result where the ID of the annotation definition is mapped onto the annotation with its associated values. The type of the values depend on the kind of annotation definition: The boolean annotation definition produces a single boolean value, the tag set annotation definition produces a list of strings representing the IDs of the selected tag options. Every value associated with an annotation can optionally have a probability (`ValueToProbability` in Figure 3.6). This is mainly used to support generated annotations through machine learning. Using the annotation generation capabilities of ActiveAnno, the `GeneratedAnnotationData` model groups generated annotations for a single project. Those generated annotations will then be transformed into annotation results by ActiveAnno. The different types of sources for annotation results are represented through the `AnnotationResultCreator` class hierarchy which allows to store annotations created by annotators, curators, annotation generators as well as imported annotations and annotation results created by the consensus of

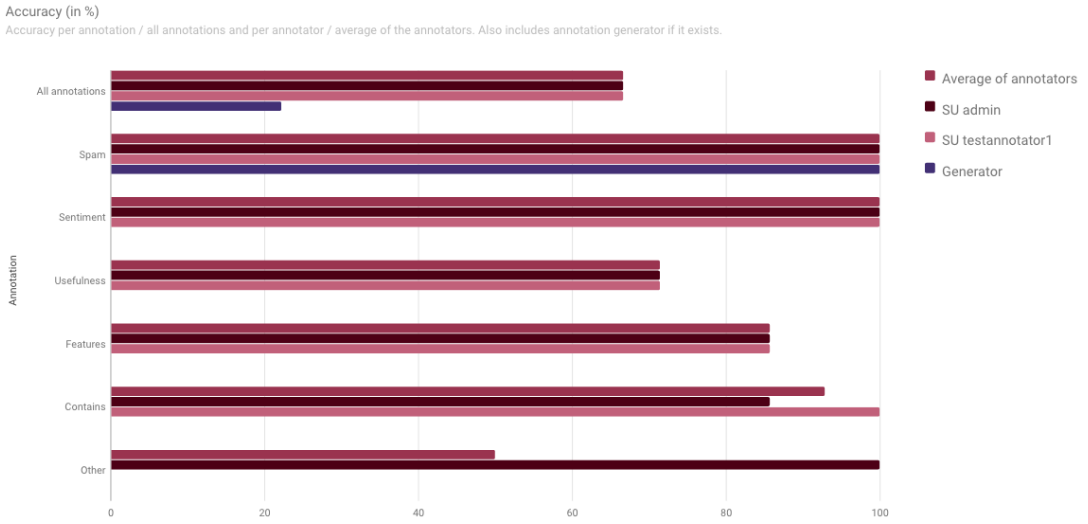


Figure 3.5: A screenshot of the analyze annotation results UI. Specifically, the accuracy chart, comparing the different annotation creators with each other for all annotations combined and each annotation individually.

multiple annotation results from different creators. Additionally, if created by a human, an annotation result will have an associated interaction log which is used to track the duration of the annotation process for the document. Finally, once conditions for finalizing an the annotation results for a document are met, the `FinalizedAnnotationResult` model stores which annotation results are part of what is considered the final or correct result. Depending on how the project is set up, this can be one or multiple annotation results. The reason for finalization is either because of a curator who created or selected the annotation result, or because the agreement policy of the project created the final annotation result automatically.

3.3.1 Annotation Result Finalization

ActiveAnno has a two-step annotation process. In the first step, annotation results are created. They can be created either by annotators, annotation generators (see Section 3.5) or they can be imported through the import annotation API (see Subsection 3.3.2). After every new annotation result, the finalization policy logic is applied to the document and its annotation results. This logic decides if the annotation process is finished for the document and a final annotation result exists. This is dependent on the project configuration. As seen in Figure 3.2, the manager of a project can set the number of annotators per document. This is the minimum number of different annotators required for each document until any additional logic is applied. For example, if the number of annotators is set to three, every document will be shown to the annotators until three have annotated the document. One annotator can only annotate every document once. After three

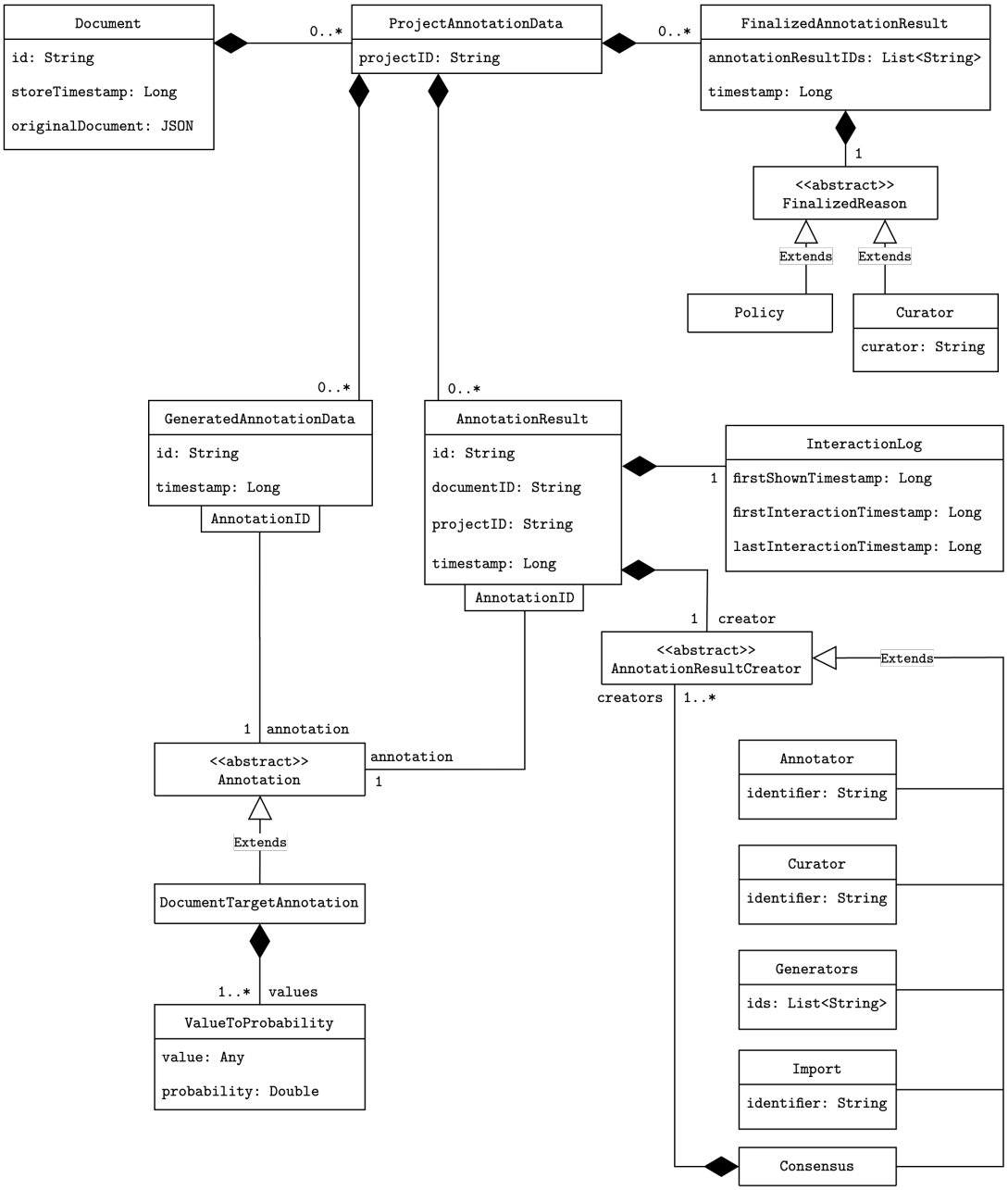


Figure 3.6: UML showing the core data models of ActiveAnno including the Document, Annotation and AnnotationResult and their relations to each other.

annotation results were created, the finalization policy is applied. This can be any of the following: Export each annotation result individually, resulting in multiple final annotation results; majority agreement per annotation, meaning there has to be at least a 51% majority of the same value for every annotation among the annotation results; majority agreement for the whole document, meaning at least a 51% majority has to exist for all annotations together; or always require a curator, where every single document has to be checked by a curator, no matter

the agreement. For the cases with agreement logic, in case of no agreement, the project can be configured to either ask an additional annotator until a majority is reached, or to ask the curator to decide. This gives project creators the ability to customize the annotation process to their needs. Using agreement on an annotation basis in combination with using more annotators as necessary might be useful in a crowd-sourcing context, while using a curator is useful if highly trained people exist in the context that can fulfill that role. In the simple case, setting the number of annotators required to one and not requiring a curator will result in every annotation result by an annotator to be instantly finalized.

3.3.2 Import and Export API

The import and export APIs are directly related to documents, annotations and annotation results. Table 3.1 shows the two endpoints of the import API of ActiveAnno. Through these, it is possible to import any JSON as a document into ActiveAnno. During the HTTP call, each document sent will get assiged a unique ID, which will be returned to the caller, and can be stored for future reference, e.g. to import annotations for the document. This is done through the second import endpoint. For it, the annotations need to conform to a map of annotation IDs (Strings) to a subclass of `Annotation` from Figure 3.6. Listing 3.1 shows an example request body for importing annotations. These annotations will be wrapped into an annotation result with the creator type `Import` (see Figure 3.6). These annotation results are then treated like annotation results from annotation generators for any further processing, meaning they can be used for pre-annotations in the UI or in the agreement logic.

Method	Route	URL parameter	Request body
POST	`/document`		Any JSON object or array of JSON objects.
POST	`/annotation/project/{projectID}/document/{documentID}`	`projectID`: The ID of a project. `documentID`: The ID of a document.	See Listing 3.1.

Table 3.1: Import API endpoints. Every route needs to be prepended by `/api/v1/import`.

Listing 3.1: Example JSON for the annotation import API. Contains two annotations, a boolean annotation for the annotation definition `SPAM` and tag set annotations with associated probabilities (which are optional) for the annotation definition `TOPICS`.

```
1 {"annotations": {
```

```
  "SPAM": {
    "target": "document",
    "values": [
        { "value": false }
    ]
  },
  "TOPICS": {
    "target": "document",
    "values": [
        { "value": "SPORTS", "probability": 0.912 },
        { "value": "TECH",   "probability": 0.722 }
    ]
  }
}}
```

Table 3.2 lists the one endpoint of the export API. Through this endpoint, annotation results can be exported into other applications for further processing. Through the GET parameters, it can be defined which annotation results will be exported. Usually, the endpoint would be called with the `since` parameter set to only get new annotation results since the last time the endpoint was called. Alternatively, if the calling application kept track of the document IDs during import, it can also set the `documentIDs` parameter to specifically get annotation results for these documents. It is also possible to configure the response structure by setting one of the Boolean flags. For example, if `includeAllAnnotationResults` is set to true, not only the finalized annotation result will be returned, but every other annotation result created during the annotation process as well.

As an alternative to the export API, every project can define a list of web hook URLs, which will be called once a document is finalized.

3.4 Annotation Definitions

Table 3.3 displays all supported types of annotation definitions in ActiveAnno, which are currently seven different types. Every annotation definition has a unique ID used for automatic handling of the annotations, e.g. as a key for exported annotations. It also has a name and short version or abbreviation of the name displayed to users in the web application. Functionally, every annotation definition defines a validation method to ensure all stored annotations adhere to the configured restrictions. Any validation errors will be propagated to the user interface to allow annotators to correct their inputs.

The first annotation definition listed in Table 3.3 is the boolean annotation definition which can be used to query a truth value from annotators. An example would be: *Is this document spam?*. As with all annotation definitions, the boolean annotation definition has the capability to express the optionality of the annotation

Method	Route	URL parameter	GET parameter
GET	`/project/{projectID}`	`projectID`: The ID of a project.	`includeUnfinished`: Boolean `includeUsedProject`: Boolean `includeDocumentData`: Boolean `includeExportStatistics`: Boolean `includeAllAnnotationResults`: Boolean `since`: Long (UTC timestamp; default null) `documentIDs`: String (Comma-separated document IDs; default null)

Table 3.2: Export API endpoint. The route needs to be prepended by `/api/v1/export`. The authentication is configurable per project (No authentication, HTTP Basic Auth, JWT). All Boolean GET parameters are false by default.

by configuring if it is optional or not. If it is, the annotator can skip creating an annotation for this annotation definition.

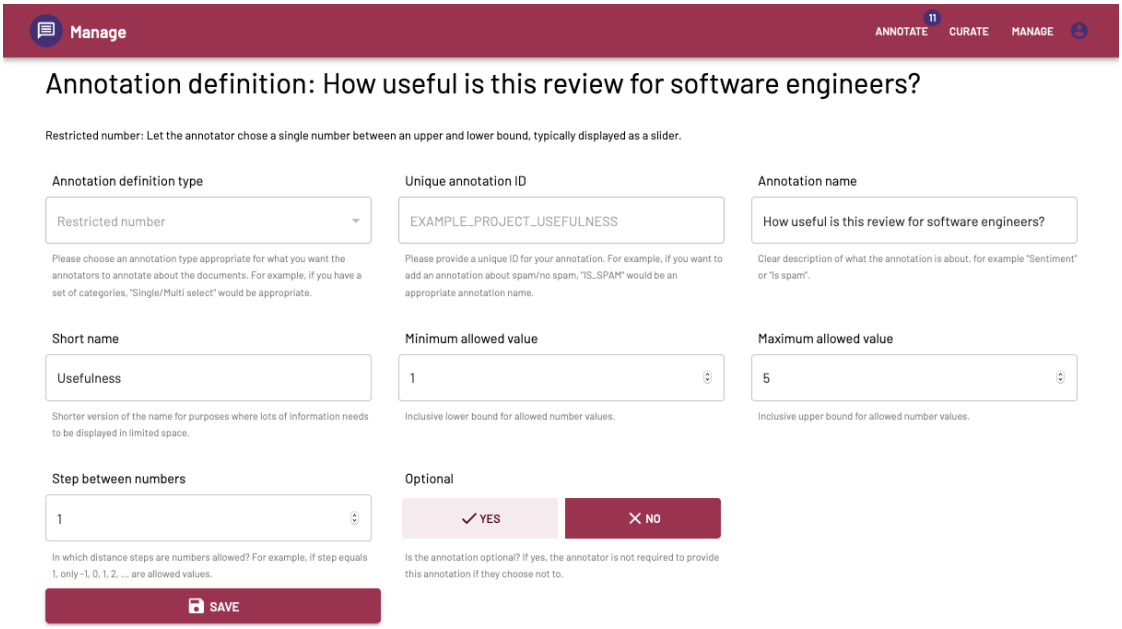


Figure 3.7: Screenshot of the editing user interface for an example restricted number annotation definition.

Three different annotation definitions exist to store number annotations. The open number annotation definition requests the annotation of any real number, which can also be optional and can be restricted to a given step size, such that values need to be expressed in steps of for example 0.1 or 1. The restricted number annotation definition additionally requires to configure a minimum and maximum

Annotation Def.	**Data type**	**Configurablility**
Boolean	Boolean	optional (true / false)
Open Number	Double	optional (true / false) step size (Double or null)
Restricted Number	Double	optional (true / false) step size (Double) minimum value (Double) maximum value (Double)
Number Range	List<Double>	optional (true / false) step size (Double) minimum value (Double) maximum value (Double)
Tag Set	List<TagSetOption>	options (List of TagSetOption) min number of tags (Int) max number of tags (Int or null)
Open Tag	List<String>	min number of tags (Int) max number of tags (Int or null) trim whitespace (true / false) predefinedTags (List<String>) use existing values as predefined tags (true / false)
Open Text	String	optional (true / false) minimum length (Int) maximum length (Int or null) document data default (String)

Table 3.3: All available annotation definition types, what data type they produce and how they can be configured.

value. The annotation value then has to be chosen between those limits. The main reason for differentiating between both cases is that numbers limited by a lower and upper bound can be annotated by using a slider, while unrestricted numbers need to be queried by providing a number input field. Using a slider might be more convenient and appropriate for some use cases, which is the reason why it is supported in ActiveAnno. An example for a restricted number annotation definition is shown in Figure 3.7, where the editing UI for annotation definitions is displayed for a restricted number. The third number-based annotation definition is the number range, which is similar to the restricted number only that is stores two number values between the lower and upper bound. This annotation can also

be produced through a slider input element by the annotator. In a context where the documents are historical texts, an example for a number range annotation definition could be *From what time in history could this document be?* with a range from the year 1500 to the year 2020 in steps of 50 years, in which case the annotator would try to narrow the range down as close as possible by providing the upper and lower bound for their estimate.

The tag set annotation definition can be used to present the annotator with a selection choice between any number of predefined tag set options, which are represented by an ID and a display name. Based on the configuration of the minimum and maximum number of required tags, the annotation definition can require a single answer (minimum and maximum set to 1), the annotation can be optional (minimum set to 0) or the annotation can be multi-select (maximum not defined or higher than 1). An example for a tag set annotation definition is *What is the sentiment of the document?* with the options *positive* and *negative*.

If the annotator is allowed to add their own tag values, the open tag annotation definition can be used. Here, the annotation definition can be configured to provide a list of predefined tags to select from, but the annotator may produce their own tags. For this annotation definition, tags are just a list of string values. It is also possible to aggregate all previously created tags for an annotation definition and add them to the list of predefined tags for the annotator to choose from. Similar to the tag set annotation definition, minimum and maximum number of required tags can be configured. An example for this annotation definition is *What are the most important topics mentioned in this news item?*. It would be possible to provide the options *sports*, *politics* and *technology* as predefined tags which would appear in the dropdown input for the annotator.

The last annotation definition supported by ActiveAnno is the open text annotation definition, where the annotator can insert a single text into a text input. For purposes of editing an existing piece of text from the original document, the annotation definition can be configured to copy the value of a field from the document data into the text input by default, allowing the user to edit that text value. An example for this would be to have an open text annotation definition for spelling error correction where the document text is copied into the input and the resulting corrected text is stored as an annotation for the original document.

The Figure 3.3 shows the annotation UI for an example project. In the *annotate document* panel, six annotation definitions are presented to the user: *Is spam*, a Boolean annotation definition; *Sentiment*, a tag set annotation definition, *How useful is this review for software engineers?*, a restricted number annotation definition with a scale from 1 to 5; *Mentioned features*, an optional open tag annotation definition with a maximum of 10 answers; *This review contains a*, an optional tag set annotation definition; and *Other things this review contains*, an optional open text annotation definition.

3.4.1 Annotation Schema and Enable Conditions

Annotation definitions are created independently from a project. They are included in a project through the annotation schema. The annotation schema puts annotation definitions in an order and associates them with project-specific configuration. They can be associated with an appropriate annotation generator (Section 3.5), and with an **enable condition**.

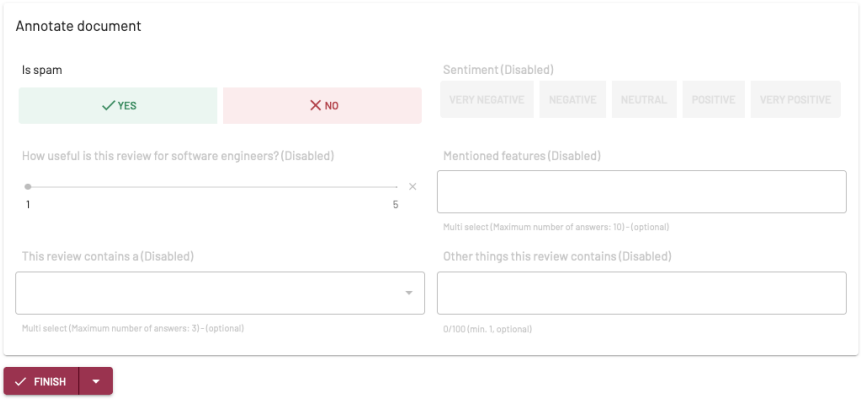
Enable conditions are a feature of ActiveAnno which allows annotation definitions to only be enabled when a certain condition is met. If no enable condition is defined, the annotation definition will be enabled by default. The first possibility of enable conditions is to make annotation definitions only enabled for a subset of documents of the project. For example, the documents for a project might have a metadata field *type* which can be one of two values. For the second type of document, it is required to annotate one additional information about the document. In this case, the project can be configured to have the annotation definition only enabled for type two and be automatically disabled for the other type.

The second possibility is to have conditions between annotation definitions. In Figure 3.8, the annotation panel is shown in two states. In the first screenshot, no annotation value is set, showing all annotation definitions except the *Is spam* annotation definition as disabled. Only when the document is annotated as *not spam* in the second screenshot, the other annotation definitions become enabled.

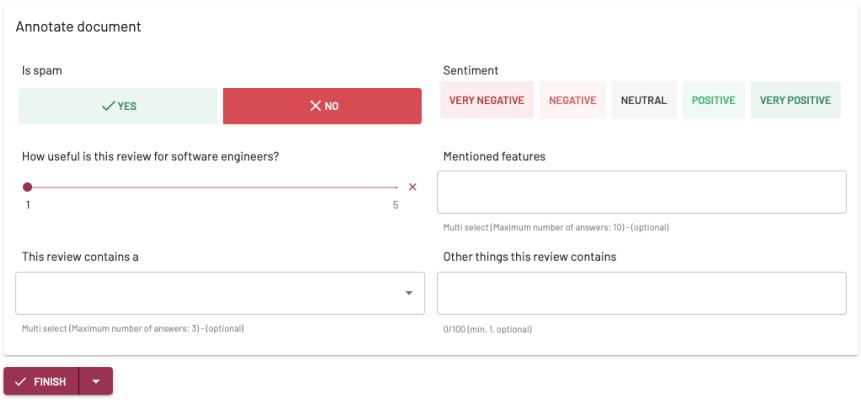
This feature has multiple benefits. First, for small differences in required annotation definitions between documents, there is no need to create multiple projects to map this behavior. Secondly, fully disabling annotation inputs prevents annotators from creating annotations where they are not required, speeding up the annotation process and more clearly communicating what is required from the annotators. And thirdly, this conditional logic is also applied when automatically creating annotations through annotation generators (Section 3.5). This means automatically generated annotations and human created annotations follow the exact same behavior, enabling a better integration of generated annotations into the remaining system.

3.5 Annotation Generator

As discussed, the integration of machine learning and the ability to automate parts of the annotation process are important to increase the efficiency of the annotation process. The basis for automation is the ability to automatically generate annotations without a human annotator. Usually, this is done through some form of machine learning. To generalize the concept and to allow for other ways to automatically create annotations, ActiveAnno defines the concept of **annotation generators**. An annotation generator is anything capable of creating an annotation for a specific annotation definition given a document and potentially other previously generated annotations for other annotation definitions. In Figure 3.9, the abstract `AnnotationGenerator` class models this through an abstract method `generateAnnotation` that needs to be implemented by every subclass. Every anno-



(a) *Is spam* not annotated



(b) *Is spam* annotated as false

Figure 3.8: Two screenshots of the annotation panel showing how inputs only get enabled once an annotation is set to trigger the enable condition.

tation generator also has to define what is the input to use for the actual annotation generation: It can be any field from the original document, or it can be any value from another created annotation that is part of the same annotation schema. This is modeled through the `AnnotationStepKey` as seen in Figure 3.9.

Currently, ActiveAnno has three inbuilt annotation generator implementations. One to automatically detect the language of the generator input using the language detection library Lingua[11]. This is an example of a statistical and rule-based annotation generator as compared to a machine learning based generator.

The second annotation generator allows to call an external machine learning service through a URL for tag set annotation definitions, which will take the response and map it into the tag set options from the annotation definition. This can be used when an already trained machine learning model exists. That model would have to be wrapped by an HTTP API to comply with the API definition of ActiveAnno for this annotation generator, which can be seen in Figure 3.10. The POST request will have the `PredictionRequest` class serialized as JSON,

[11] `https://github.com/pemistahl/lingua`

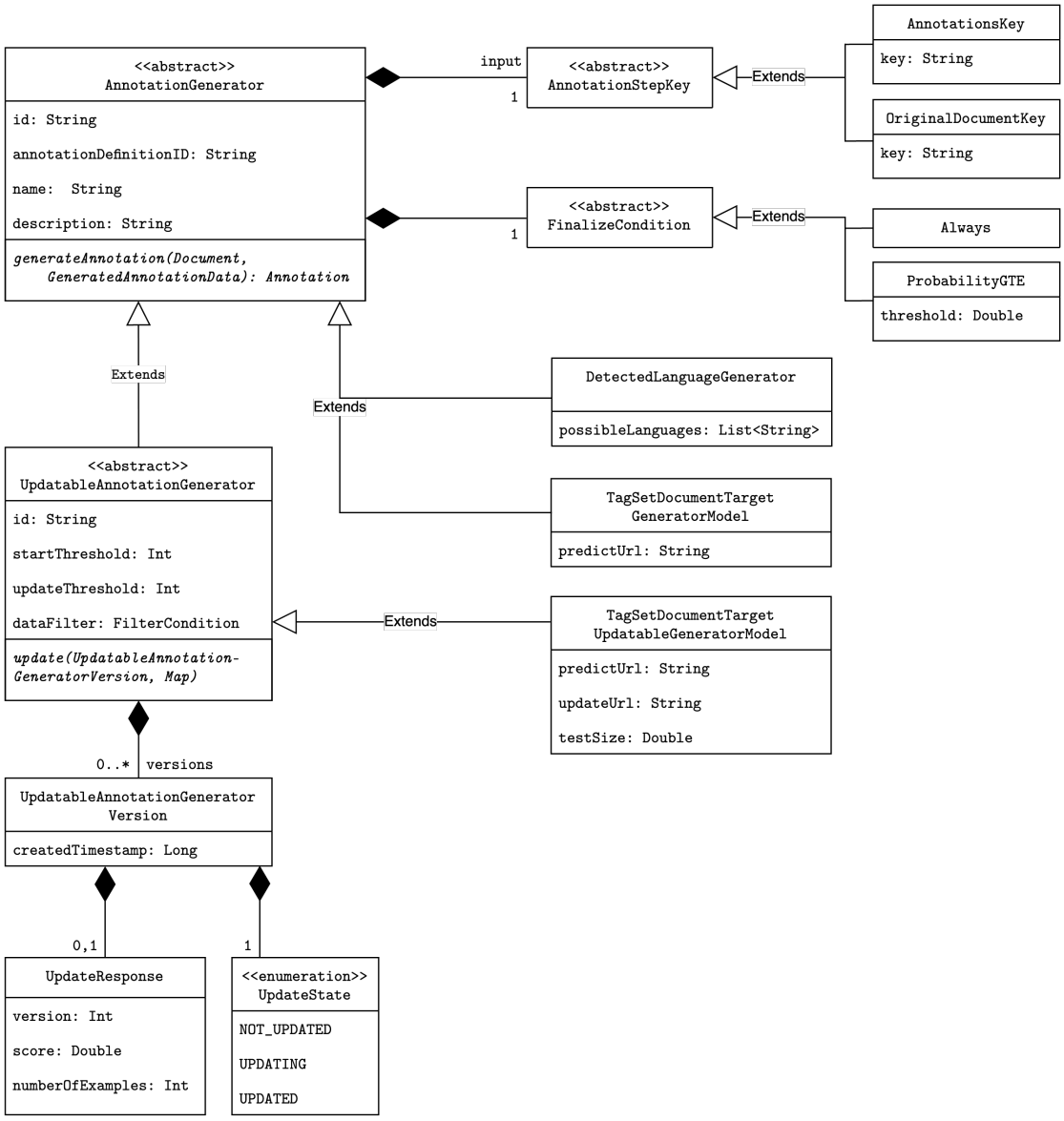


Figure 3.9: UML showing the classes related to the annotation generator features of ActiveAnno.

and the response should be a JSON array which will be deserizalized into a `List` of `PredictionResponse` objects, which contain the original information from the request and the associated values with their probabilities, which are optional. As seen in the UML diagram, the API is structured to allow for multiple documents to be predicted at once.

The third annotation generator is similar to the second one, but also supports automatically updating the external machine learning model by sending an HTTP request with the training data in the body. The POST request will be a JSON body serializing the class structure from Figure 3.11. To support this functionality, the concept of an **updatable** annotation generator exists. This kind of generator ex-

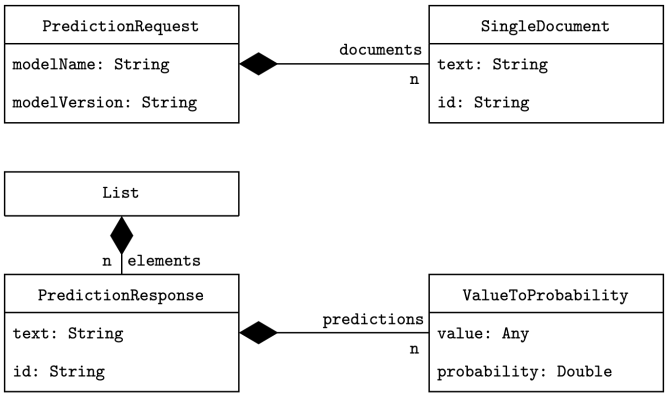


Figure 3.10: UML diagram of the API of the inbuilt tag set annotation generators to allow an external machine learning model to provide predictions.

tends the base annotation generator, but also requires its subclasses to implement an `update` method (as seen in Figure 3.9), where the training data will be aggregated and used to train or update the annotation generator. For this, updatable annotation generators also need to define a threshold when to start the training and when to update an existing model. For example, the first model should be trained after 100 training samples exist, and then it should be updated for every 25 new training samples. An updatable annotation generator is versioned, where every version has a version number and an update state, to ensure the version is actually usable for generating annotations. This can also be seen in Figure 3.9 where the versioning is modeled through the `UpdatableAnnotationGeneratorVersion` class.

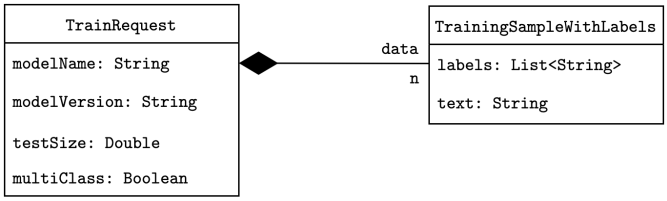


Figure 3.11: UML diagram of the API of the inbuilt updatable tag set annotation generator to train or update an external machine learning model.

3.5.1 Generator API

Table 3.4 lists the endpoints of the generator-related API. The first endpoint allows to trigger the generation of annotations for all annotation generators of every project. Annotations will be generated either because there are no generated annotations for the project and generator yet, or because the generator is *updatable* and a new version of the generator exists. This behavior can be configured in the project setup. The second endpoint has the same behavior as the first one, but for one specific project.

Method	Route	URL parameter	GET parameter
GET	`/generateAnnotations`		limit: Int (default null) chunkSize: Int (default 100)
GET	`/generateAnnotations/project/{projectID}`	projectID: The ID of a project.	limit: Int (default null) chunkSize: Int (default 100)
GET	`/update`		
GET	`/update/{generatorID}`	generatorID: The ID of an annotation generator.	

Table 3.4: Generator API description. Every route listed needs to be prepended by `/api/v1/generators/`.

The third endpoint can be called to update all updatable annotation generators. Every generator checks itself, if it can be updated already, based on the defined thresholds for new samples. The fourth endpoint provides the same behavior, but for a specific generator.

To make sure annotations are generated and updatable annotation generators are updated regularly, these endpoints should be called periodically, for example through a job scheduler like cron. The reason that this behavior is wrapped behind an API instead of just running it periodically inside the application itself is to give the users of ActiveAnno full control of when these tasks should be executed.

3.5.2 Project Machine Learning Integration

After creating the annotation definitions and annotation generators and after providing an external machine learning service in compliance with the API of ActiveAnno, the last step to integrate machine learning into ActiveAnno is to define how the generated annotations should be treated. For this, a project has multiple configuration possibilities.

The first one is the handling policy of generated annotation results. This policy can be set to ignore the results. Then, they will only be displayed to curators but not actively integrated into any further application logic. It can be set to use the generated annotations as pre-annotations for the annotators. In this case, the annotations will fill out or select the inputs in the annotation panel, giving the annotators the option to just accept the automatically generated annotations, which can reduce the annotation effort. Lastly, it is possible to set the generator results to be treated equal to an annotator. In this case, the results will be included in the finalization logic. The `FinalizeCondition` from Figure 3.9 is used in this case to decide if a generated annotation is good enough to be used in the finalization logic. Usually, this means setting a confidence threshold as the finalize condition,

for example a required confidence of 80% or more.

The other important configuration is the sorting policy. With regards to generated annotations, it is possible to overwrite the normal sorting order of the project. This can be set to prefer documents with existing generated annotation results first. In this case, if only a subset of documents have received their generated annotations at a point in time, they will be preferred in sending them to the annotators. This means that if pre-annotations are available, they will always be shown before documents without pre-annotations. The second option is to set the sorting to *active learning with uncertainty sampling*. This is used for supporting active learning, where the documents with the lowest confidence values associated with the generated annotations get presented to the annotators first. The active learning process with ActiveAnno will be examined in more detail in Subsection 3.5.3.

There is a completely alternative approach for machine learning integration available: Through the import annotation API (see Subsection 3.3.2), imported annotations can be used instead of internally generated annotations for all the same purposes. For updating a machine learning model, the REST API or webhook support can be used to get the final annotation results. In this case, all the logic regarding how to extract the data and when to update the model need to be implemented externally. This approach might be more useful if the required logic or process is vastly different from the inbuilt annotation generator concept.

3.5.3 Active Learning Process

When there is no existing training data for an annotation definition, ActiveAnno can be used to build up a training dataset based on active learning with uncertainty sampling. Starting with a setup of the annotation definition with an associated updatable annotation generator created inside ActiveAnno, both should be integrated into a project. The training data can either be imported once at the start of the active learning process, or continuously, for example if the data comes from a news crawler or the Twitter API. The active learning process would work as follows: First, when there is no training data, documents get manually labeled by an annotator. If the start threshold of the annotation generator is reached, calling the update generator API (Subsection 3.5.1) will trigger the training data to be aggregated and sent to the external service. After that, calling the generate annotations API will result in the newly trained generator model to create predictions for all remaining documents for the project. These will be stored and when the projects sorting policy is set to active learning, the confidence values from the newly generated annotations will be used to sort the documents to annotate with the lowest confidence documents being annotated first.

After some amount of those documents were annotated by an annotator, the update generator API and generate annotation API can be called again to update the model and all the predictions with the now larger dataset with training samples selected by least confidence. This process can then be repeated until the machine learning model is performing well enough to be partly or fully automated. This

is similar to the "tag-a-little, learn-a-little loop" from MAT [1]. If combined with enabling annotations, it is also very similar to the annotation process of RapTAT [6].

To partly automate the process, the project has to be configured to treat the generator as an annotator and to require one annotator per document. Additionally, the finalize condition has to be set to some confidence threshold, for example 80%. Then, only the documents with a confidence value below 80% will be required to be annotated by hand. To fully automate the task, set the finalize condition to *always*. Then there is no condition and annotations will be accepted automatically.

Chapter 4

Experiments

To test ActiveAnno in an industry setting while gaining relevant research insights by investigating the two research questions in Section 1.2, two experiments were conducted in partnership with a local company. Basis for the experiments are real-world use cases and a dataset from the company. The experiments are designed in a way to give both quantitative and qualitative results for using ActiveAnno in a specific industry context. For the experiments, ActiveAnno was deployed to the existing service cluster and configured using environment variables.

4.1 Machine Learning Integration

For the machine learning integration, a closed-source service was deployed, which wraps the fastText [9] Python library[1] with a thin HTTP API. It supports the text classification through supervised learning as provided by fastText. The service conforms to the API specified by ActiveAnno as described in Section 3.5.

The fastText API allows for some hyperparameters to be set[2], for example the learning rate, the number of dimensions, the number of epochs or the size of word n-grams. Those hyperparameters were never changed during the experiments as to not introduce any unwanted variability and were set to a learning rate of 0.1, dimensions of 200, epochs of 50 and word n-grams of 3.

4.2 Data Preparation

Basis for all experiments are textual comments[3] given by real humans through a software provided by the industry partner of this thesis over the last several years. There is variability in the language, the context of the comment and the question asked to which the comment is the answer over the initial set of about 400,000 comments. To reduce unnecessary complexity and streamline the dataset, the comments used for the experiments were restricted to comments of German

[1] `https://github.com/facebookresearch/fastText/tree/master/python`

[2] `https://fasttext.cc/docs/en/options.html`

[3] In this chapter, *comment* and *document* are used interchangeably, as a comment is a specific type of document in this context.

language given in a retail shop context to a question about general, open-ended feedback. This limited the number of comments to 250,914. The comments have an average length of 34.1 characters and a median length of 20 characters with the minimum length being 1 character and the maximum length being 2607 characters. These short comments make document-level annotations generally more applicable as compared to longer text documents, as the benefit of specifying annotations on a span level compared to the additional effort is lower. Most comments express a single thought with a single topic and a single sentiment, though there are enough outliers that make the annotation process more complex than that. The most common 100 comments make up 13.62% of the total of 230,421 comments while 67.7% of those comments occur only once and are therefore unique. Table 4.1 shows the ten most common comments in the dataset while Table 4.2 shows some example comments with lower frequency to give an impression of what the dataset looks like.

Original Comment	**Translated Comment**	**Count**
Nein	No	3137
Gut	Good	1539
Danke	Thanks	1150
Super	Great	1120
Hallo	Hello	1063
:-)	:-)	1016
Weiter so	Keep it up	961
Hi	Hi	943
Alles super	Everything great	880
Cool	Cool	818

Table 4.1: The ten most common comments, their English translation and their number of occurrences in the dataset.

First, the documents were split between the two experiments. The first experiment needs a lot more data that is used to train machine learning models upfront. Therefore, the documents were randomly split 90% to Experiment 1 and 10% to Experiment 2. Table 4.3 shows the distribution of documents between the experiments. For Experiment 1, 20,493 comments were discarded because they were too recent and might have been seen already by the annotators of the experiment. All data preparation was done externally in a script with a connection to the preexisting database. The comments were then imported through the document import API (Subsection 3.3.2) of ActiveAnno. For Experiment 1, 144,648 comments with existing annotations in the preexisting database were randomly chosen as training data for the annotation generator in form of fastText models. The annotations were converted into the appropriate format for ActiveAnno and imported through the annotation result import API. During the import, the project of Experiment

Original comment	**Translated comment**
Gute Beratung, freundliches personal	Good advice, friendly staff
Weiter so.!!!!!	Keep it up.!!!!!
Hgjhihtz hgjgijb	Hgjhihtz hgjgijb
Unerträglich die Temperatur und die Luft. Einfach zum umfallen	The temperature and the air were unbearable. Just to fall over
Waren alle sehr nett	They were all very nice
Uhhhjkkhghkl	Uhhhjkkhghkl
Perfekter Einkauf Beratung Freundlichkeit Kasse super !!!!	Perfect shopping advice friendliness checkout great !!!!
shrsjtehrs	shrsjtehrs
Ihre Musik geht gar nicht :-(	Your music is a no-go :-(
Mehr Auswahl für Kinder	More selection for children

Table 4.2: Ten uncommon example comments that occur less than ten times each and their translation into English.

1 was set to require one annotator per document and finalize annotation results without a curator, resulting in the imported results and the associated documents to be finalized immediately and therefore being available as training data for the annotation generators.

	Total	For annotators	For generator	Discarded
Experiment 1	225,874	60,733	144,648	20,493
Experiment 2	25,040	25,040	0	0

Table 4.3: Distribution of documents between experiments.

4.3 Experiment 1: Influence of Pre-Annotations on Quality and Efficiency

The first experiment is a controlled field experiment making use of preexisting training data in the industry setting to answer Research Question 1 about the effect of pre-annotations in the form of generated document-level annotations by a machine learning model on the quality of annotations and the efficiency with which the annotations were created by the annotators.

By comparing two conditions, one with pre-annotations and one without, the goal is to find out how much faster the annotators are when presented pre-annotations, and if pre-annotation has an influence on the inter-annotator agreement or especially the accuracy of the annotators in a statistically significant way. Secondary

goals are to demonstrate how ActiveAnno can be used to conduct a controlled experiment and how ActiveAnno is able to implement a typical annotation setup in an industry context.

4.3.1 Project Setup

To conduct the experiment, a new annotation project was created inside the deployed ActiveAnno instance. The number of annotators required for each document was set to two, such that average accuracy and time statistics can be calculated over both annotators to get more meaningful results. The finalization policy of the project was set to always require a curator for each document, regardless of annotator agreement. Through this, a gold standard is created by the curator, which can be used to evaluate the annotator and generator accuracy.

The layout of the annotation view was minimal, showing only the document text, the associated question asked to which the text is the answer, and an number used for communicating the annotators progress. While there is more metadata available which could be used to help the annotators make their decisions, it was not included for the experiment, because this metadata is also not available to the machine learning models. Adding more metadata would make comparisons between the annotators and models less appropriate.[4]

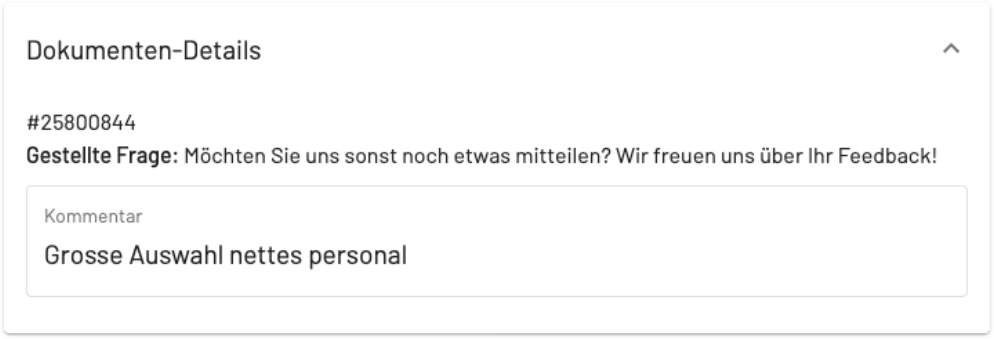


Figure 4.1: Screenshot of an example document of the experiment displayed in ActiveAnno. The question translates to “Do you have any comments for us? We appreciate your feedback!” and the comment translates to “Large selection friendly staff”.

Annotation Definitions

The annotation definitions for this experiment are based on existing annotation definitions already used in the industry setting. They were also used for the experiment in order to utilize the existing annotations as training data and because using a real-world annotation setup can give more insights into typical requirements for the annotation process compared to an artificially manufactured setup.

[4]It is future work to allow the integration of metadata into the annotation generator process.

The first annotation definition is the approval of a comment, a binary classification task. It is defined as a tag set annotation definition with two options, *approve* or *not approve*. The goal of this annotation is to filter spam as those comments are not relevant for any further analysis of their content.

The second annotation definition is the sentiment of a comment. It is also defined as a tag set annotation definition and has three options: *positive*, *ambivalent* and *negative*. As the annotation is targeted on the document level, the option *ambivalent* is necessary to describe comments with multiple sentiments. As a multi-class classification task, exactly one of the three options is required to be selected by the annotators.

The third annotation definition are the topics or categories of a comment. There exist eight context-specific categories which describe distinct, common topics of a comment. While the actual categories had to be disguised, an analogous example would be news items categorized by their general topic, such as sports, politics or tech. Categories are a tag set annotation definition in form of a multi-labelled classification task. This increases the complexity of the annotation and getting the whole annotation correct means deciding for each option if it should be selected for a comment. A minimum of one category has to be selected.

The project makes use of the *enable condition* feature (see Subsection 3.4.1). The sentiment and category annotation definitions are only required if the approval annotation equals *approve*, because spam comments have no sentiment or category anyway. As described in Subsection 3.4.1, this means the sentiment and category inputs are disabled in the UI as long as the approval annotation definition was not set to *approve*. This can be seen in Figure 4.2. Equally, the annotation generators will only generate values for the sentiment and categories if the annotation generator predicts the value *approve* for the approval annotation definition.

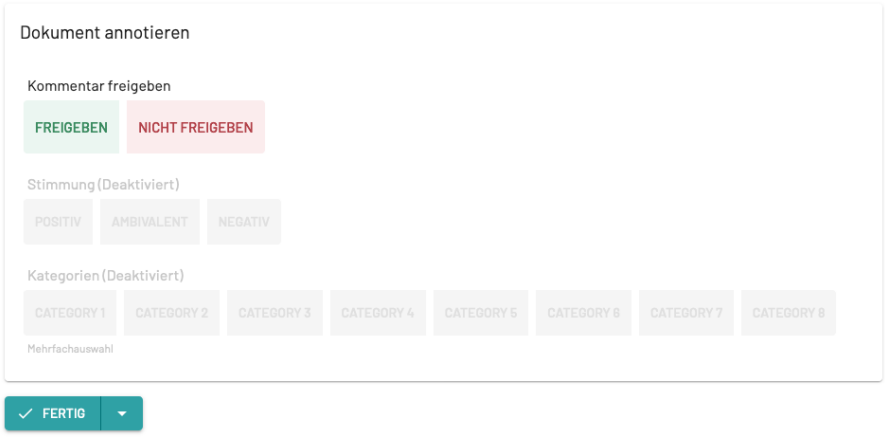
For the experiment, the enable conditions between the annotation definitions and the multi-labelled annotation definition *categories* add some complexity to the task, because not every annotation result will need the same number of button clicks to be finished and not every annotation result will have the same number of annotations. Some results will have only one annotation, *not approved*, and others will have values for all three.

Annotation Generators

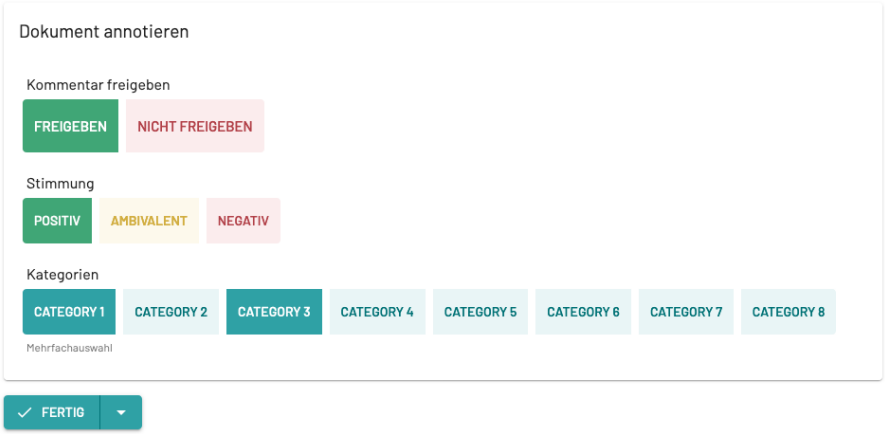
Each annotation definition has an associated annotation generator, which allow the training of a machine learning model and prediction by that model though calling an external service over HTTP. As described in Section 4.1, the external service uses fastText to make text classification.

For the experiment, all three models were trained only once before the experiment, by calling the ActiveAnno update generator API (Subsection 3.5.1). All models have a size of about 1.7GB and the training time took between 79 and 98 seconds on a virtual machine with 3GB RAM and 1.25 CPU cores (2.6GHz).

The annotation generator model for the *approval* annotation definition was trained with 144,648 samples of which 25% were used for testing, leaving 108,486



(a) Before annotating



(b) After annotating

Figure 4.2: Screenshot of the annotation panel before and after doing the annotations (category names are disguised).

samples for training and 36,162 for testing. The model has a test accuracy of 0.9.

Because of the conditional nature of the sentiment and categories annotation definition as described in 4.3.1, only a subset of all 144,648 comments actually have a sentiment and categories annotated. For the sentiment model, this resulted in 79,799 train samples and 26,600 test samples with a test accuracy of 0.9. For the multi-labelled categories classification task, fastText was configured to use the *one vs. all* loss. Because of some inconsistencies in the dataset, there were more samples for the categories model than for the sentiment model: 89,096 train samples and 29,699 test samples. With a precision of 0.72 and a recall of 0.63, the F1 score for the categories model is 0.67. This is a considerably worse performance as compared to the other two annotation definitions, which correlates with a lower inter-annotator agreement for this annotation definition as well, as will be shown later.

4.3.2 Experimental Design

The annotation process for the experiment has two subjects, the annotators. As described, both annotate the same randomly sampled documents. The independent variable for this experiment is whether the buttons in the user interface are preselected through machine learning generated annotations or not, which results in two different conditions. The dependent variables are measured directly by ActiveAnno: The duration from when the document is first shown the annotator until the *finish* button was successfully clicked and all required annotations were made; the duration from the first button click to the last button click, called the interaction duration; the correctness of the annotation results which is calculated by comparing them to the gold standard created by the curator; and the agreement between the annotators.

The assignment of conditions was done in a within-subjects design. Each subject or annotator was assigned each condition, which reduces the possible negative impact of inter-subject differences on the results. The risk of learning effects was mitigated through two measures: Both annotators were made familiar with the tool and experiment setup through an instruction and test meeting previous to the actual experiment. Also, condition order was switched between the two days of the experiment.

4.3.3 Execution

After setting up the annotation project and the machine learning models for the experiment, the models predicted the annotations for all documents of the experiment which were not already part of the training data. Due to this, generated annotations exist for both conditions to compare them to the annotators as well as the gold standard, even though they are only visible as pre-annotations in one of the conditions.

Prior to the first day of the experiment, an instruction meeting between the author of this thesis and the two annotators was conducted over video chat. It was used to explain the process of the experiment, to make sure the web application works for both annotators and to do a test run of about 50 documents such that the result analysis could be tested and checked for any possible bugs. The experiment itself was done over two days, taking 45 minutes on the first day and 1 hour on the second day. It was done over video chat with both annotators at the same time, trying to get as close to a usual office situation. Both annotators used the same laptop model, with annotator 1 (A1) using a mouse and annotator 2 (A2) using a trackpad throughout the whole experiment. After communicating the start of the experiment, all participants in the video chat were muted during the annotation process to prevent unwanted influences. The number of annotated documents were tracked in order to get close to equal number of documents per condition. After about half of the time on each day, the annotators were informed about the switch to the other condition. This was done by changing the project configuration from ignoring generated annotation results to using them as pre-annotations. On the first

day, the no pre-annotations condition was first and the pre-annotations condition second. This was swapped on the second day.

On both days, after the annotation process was done, a curator controlled every single document and the annotation results of A1, A2 and the generator to decide on the correct annotations. This created the gold standard used in the further analysis.

4.3.4 Results

In total, 944 documents were annotated by both annotators and the curator to create the gold standard. Of those, four documents were removed upfront as they were outliers with very high duration statistics. This left 940 documents with 465 for Condition 1 without pre-annotations (C1) and 475 for Condition 2 with pre-annotations (C2).

There was another unfortunate source of outliers, created by the categories annotation generator. The fastText model was trained with the one vs. all loss, which results in a binary decision if the category is selected for every category independently. One resulting edge case is the possibility of the model not predicting any category, when every single one vs. all probability is below 0.5. Having no category was not defined as an option beforehand, as the minimum number of categories for annotators was set to 1. This occurred 53 times in total, 30 times for Condition 1 and 23 times for Condition 2. The comments with missing predictions resulted in no pre-annotations for the category annotation definition. Therefore, they could not be included for the pre-annotation condition. They were also removed from the other condition, because by their nature of being hard to predict for the model, they were also hard to correctly annotate for annotators. One side effect of the removal is that the generator accuracy was inflated: In the original 940 documents, the average accuracy for the categories generator is 74.21%, in the smaller dataset of 887 documents it is 81.39%. For every other further metric, the dataset with 887 documents is used.

Annotation Duration

	A1	A2	Avg.
C1 and C2 (n=887)	2.71	3.31	3.01
C1 (n=435)	3.27	3.74	3.51
C2 (n=452)	2.17	2.88	2.53
Difference C1-C2	1.10	0.86	**0.98**

Table 4.4: Full annotation duration in seconds for annotator 1 (A1), annotator 2 (A2), Condition 1 with no pre-annotation (C1) and Condition 2 with pre-annotation (C2).

Table 4.4 show the full annotation duration from the moment the document is

displayed to the annotator to the moment the *Finish* button is pressed successfully. Overall, A1 is faster than A2 across the board, with a 0.6s faster average for all 887 documents. This might be just because A1 is faster in general, but could also be explained by the fact that A1 used a mouse while A2 used the trackpad of the laptop for the experiment.

Comparing both conditions, the expected result of a shorter annotation duration for the condition with pre-annotation occurred with an average of 0.98s faster annotation time for the pre-annotation condition. This is 27.92% faster for C2 compared to C1.

A potential source of noise in Table 4.4 is that not every document has the same number of annotations due to the conditional dependencies between the *approval* annotation definition and the other two annotation definitions. When normalizing over the number of annotations, C2 is still 26.45% faster with 1.55s per annotation for C1 compared to 1.14s per annotation for C2.

	A1	A2	Avg.
C1 (n=435)	1.83	2.17	2.00
C2 (n=452)	0.27	0.33	0.30
Diff. C1-C2	1.55	1.84	**1.70**

Table 4.5: Interaction duration in seconds for annotator 1 (A1), annotator 2 (A2), Condition 1 with no pre-annotation (C1) and Condition 2 with pre-annotation (C2).

The second duration statistic tracked by ActiveAnno is the interaction duration from the first to the last click or keystroke per document. Table 4.5 shows the interaction duration between both conditions. For C1, the average interaction duration is 2s, meaning that of the average full duration of 3.51s as displayed in Table 4.4, 1.51s on average are spent reading the comment before creating the first annotation through a button click, and 2s are spent making the annotations and pressing the *Finish* button. In comparison, C2 has an average interaction duration of 0.3s, because most documents only receive a single press of the *Finish* button, resulting in an interaction duration of 0.

Inter-Annotator Agreement

The initial hypothesis is that pre-annotation would increase inter-annotator agreement, because the annotators are presented the same pre-annotation values and might be enticed to trust the generated annotations in cases where they are unsure themselves. Table 4.6 shows the inter-annotator agreement for both conditions. While there is a slight increase in agreement across the board, it is not statistically significant. When calculating the p-value for the one-tailed hypothesis that agreement increases, the result for p is 0.11 for all annotations, 0.16 for the approval annotation definition, 0.49 for the sentiment annotation definition and 0.12 for the

	All annotations			Approval			Sentiment			Categories*		
	n	#	%	n	#	%	n	#	%	n	#	%
C1	435	390	89.66	435	415	95.40	273	255	93.41	273	233	85.35
C2	452	416	92.04	452	437	96.68	275	257	93.45	275	244	88.73
Diff.			-2.38			-1.28			-0.05			-3.38

Table 4.6: Inter-annotator agreement between Condition 1 without pre-annotation (C1) and Condition 2 with pre-annotation (C2) for all annotations and each annotation individually. For each, the table shows the total number of documents (n), the number of agreements (#), and the percent agreement (%). *Note that as described before, 53 documents were excluded from the analysis because of no generated annotation for the categories annotation definition. When not removed, the inter-annotator agreement for categories is 65.85% for C1 and 67.12% for C2.

categories annotation definition. When defining the significance level at 0.05, none of these results are statistically significant.

Accuracy

All ann.	n	# A1	% A1	# A2	% A2	**% Avg.**	# Gen.	% Gen.
C1	435	408	93.79	402	92.41	**93.10**	370	85.06
C2	452	430	95.13	424	93.81	**94.47**	390	86.28
Diff C1-C2			-1.34		-1.39	**-1.37**		-1.23

Table 4.7: Number of correct annotations (#) and percent accuracy (%) for Condition 1 without pre-annotation (C1), Condition 2 with pre-annotation (C2) and annotator 1 (A1), annotator 2 (A2), annotator average (Avg.) as well as the generated annotations (Gen.) over all annotations.

Approval	n	#A1	%A1	#A2	%A2	**%Avg.**	#Gen.	%Gen.
C1	435	425	97.70	419	96.32	**97.01**	416	95.63
C2	452	447	98.89	438	96.90	**97.90**	434	96.02
Diff C1-C2			-1.19		-0.58	**-0.89**		-0.39

Table 4.8: Number of correct annotations (#) and percent accuracy (%) for the approval annotation definition.

Tables 4.7 - 4.10 show the number of correct annotations and accuracy results for all annotations and each annotation. There was no statistically significant impact of pre-annotation on the accuracy of the annotators. When calculating the p-value for the two-tailed hypothesis that accuracy could increase or decrease with

Sentiment	n	#A1	%A1	#A2	%A2	**% Avg.**	#Gen.	%Gen.
C1	273	264	96.70	256	93.77	**95.24**	256	93.77
C2	275	265	96.36	260	94.55	**95.45**	254	92.36
Diff C1-C2			0.34		-0.77	**-0.22**		1.41

Table 4.9: Number of correct annotations (#) and percent accuracy (%) for the sentiment annotation definition.

Categories	n	#A1	%A1	#A2	%A2	**%Avg.**	#Gen.	%Gen.*
C1	273	251	91.94	240	87.91	**89.93**	221	80.95
C2	275	257	93.45	248	90.18	**91.82**	225	81.82
Diff C1-C2			-1.51		-2.27	**-1.89**		-0.87

Table 4.10: Number of correct annotations (#) and percent accuracy (%) for the categories annotation definition. *Without removing the 53 outliers, the generator accuracies are 72.94% for C1 and 75.50% for C2.

pre-annotation, the smallest p-value from all annotations is 0.34 for the approval annotation and the average of both annotators.

Unstructured Interview with Annotators

In addition to the quantitative results, there was also qualitative feedback gathered through an unstructured interview after the experiment. It was a short interview asking for general feedback about the tool and their feelings about both conditions.

A1 stated that they finished a document too fast two or three times in Condition 2 with pre-annotation. Specifically, they wanted to add another category for the categories annotation definition, but clicked too fast and wanted to correct their decision afterwards. A1 stated that in that case, a back button would have been useful.

A1 also stated that they felt less concentrated at the end of the second day of the experiment, after annotating documents for approximately 1 hour. They felt like having to read a comment more than once to understand it.

A2, who was using a trackpad during the experiment, stated that after a while it felt exhausting to use the trackpad to click the buttons in the condition without pre-annotation.

In general, both annotators gave positive feedback about the annotation process using ActiveAnno compared to the previously existing internal annotation tool. They stated that it was easy, fast and satisfying to use.

4.3.5 Discussion

The experiment could not find any statistically significant impact of pre-annotation on neither inter-annotator agreement nor annotator accuracy. At the same time,

the annotation duration decreased by 28% when having pre-annotations for the annotators. Having pre-annotations reduces the physical effort of clicking the input buttons: the user can hover the *Finish* button and only needs to click the actual annotation input buttons if the pre-annotation is incorrect. The pre-annotation might also reduce the mental effort by changing the task for the annotator of coming up with annotations to the task of reviewing the generated annotations and only correcting when necessary.

Even with no statistically significant differences in accuracy, as stated by A1, there is still the possibility of misclicks or clicking the *Finish* button too fast. Adding a revert or back action functionality, which is currently not supported by ActiveAnno, can be used to prevent those cases.

While clicking too fast might be a potential source of wrong annotations, so is a lack of concentration. Both A1 and A2 communicated forms of exhaustion or low concentration after a longer period of annotating. Pre-annotation can noticeably decrease overall annotation time and therefore might prevent errors made due to low concentration. It also appears to be useful to split longer annotation tasks over multiple sessions to prevent loss of concentration.

Regarding the ability of ActiveAnno to be used for a real-world annotation setting, Subsection 4.3.1 documented how such a setting could be expressed in ActiveAnno. The positive feedback from the annotators about the annotation process in ActiveAnno is an indication that ActiveAnno provides a good user experience, though another experiment setup would be required to gain quantitative information about its usability. This chapter also documented how ActiveAnno was used to conduct the experiment, showing its capabilities for conducting controlled experiments. There was an explicit effort made by the author that all analysis was directly programmed into ActiveAnno in a configurable way, and that a data export for the results is available directly in the user interface as well for any further calculations.

Threats to Internal Validity

The goal of the experiment was to accurately measure annotation time while conducting a field experiment with as close to normal work conditions as possible. The trade-off made was to be present as the conductor of the experiment during the annotation process, making sure no other work tasks or private messages etc. interfered with the annotation process. This might have inflated the concentration and effort and therefore the accuracy of the annotators, which could have had an disproportionate on Condition 2. To minimize the negative effects, everyone in the video call was muted during the actual annotation task. Also, as the annotators are generally well-trained, it is likely they would make an effort to produce correct annotations anyway. This is likely to increase the focus and effort of the annotators which might inflate their accuracy. To minimize this effect, the amount of communication during the experiment was minimized.

Threats to External Validity

Even though a considerable effort has been placed to make sure the internal validity is high, the scope of the experiment was limited in nature. The experiment was done with well-trained annotators and the task was very similar to their usual job. The pre-annotations were generated by a well-performing machine learning model with a large training set. The complexity of the annotation task itself could have also been higher, with different types of annotation definitions, like numbers, open tags or texts. As was shown, the most complex annotation definition, *categories*, performed worse in accuracy for annotations and machine learning model alike, compared to *approval* and *sentiment*. Therefore, the results might not generalize for tasks where well-performing machine learning models are not easily achievable. Also, using less qualified annotators with potentially bad incentives like pay per finished annotation might abuse pre-annotations. This is a problem in general, not specific to pre-annotation cases. Detecting bad annotators is a requirements for such situations.

4.4 Experiment 2: Comparing Active Learning Approaches

While the first experiment used the machine learning integration of ActiveAnno to analyze the benefits of pre-annotations on annotation efficiency, the second experiment investigates the effects of active learning as another approach to optimize efficiency. This is done by answering Research Question 2 about how active learning with uncertainty sampling compares to incremental learning with random sampling with regards to the generated annotation quality given a new machine learning model without preexisting training data (Section 1.2). This question shall be answered in the context of using ActiveAnno with its active learning integration to gain additional insights about how such a setup would be done in practice.

4.4.1 Project Setup

For this experiment, another project was created in ActiveAnno. It requires one annotator per document for the annotation result to be considered finished. That annotation result is also accepted as the final result without a curator. The author of this thesis is the annotator for this project.

As described, the documents for this experiment were chosen randomly from the whole set of documents and are randomly sampled for each condition. The layout for displaying the document is the same as in Experiment 1, which can be seen in Figure 4.1.

Annotation Definition

The annotation definition for this task was created based on an industry requirement to get more information about how useful a comment is to read for the target

audience. When filtering out spam, there still are lots of comments with low information value, which are not spam but also might not be important enough to be shown in specific contexts. Therefore, a new tag set annotation definition is defined about the *value* of a comment. There are four tag set options: A comment can be *very useful* to read, meaning it provides deep insights showing a complex thought by the comment author; it can be *useful* to read, meaning it provides value that cannot be expressed through just using the sentiment and category of the comment; it can be just *okay* in which case it might be just displayed in a chart of sentiment and category; or it can be *spam*, in which case it would be completely ignored for further analysis. Table 4.11 shows one example comment for each option and Figure 4.3 the interaction UI for the annotation definition.

Option	**Original comment**	**Translated comment**
Very useful	Sehr gute neue Gestaltung der Innenräume, gute Übersicht durch neue Ständer, und nicht zu überladen. Bravo, wir kommen wieder.	Very good new interior design, good overview with new stands, and not too overloaded. Bravo, we'll be back.
Useful	Bücher bitte besser anordnen. Danke	Please arrange books better. Thank you
Okay	Alles super gut	Everything was super good
Spam	Ihjhjikj	Ihjhjikj

Table 4.11: One example comment with an English translation for each tag set option for the *value* annotation definition.

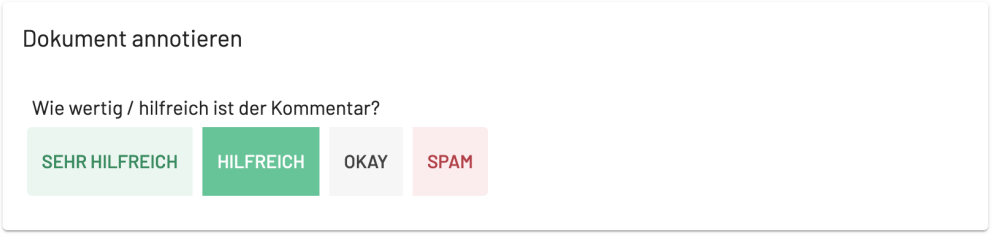


Figure 4.3: Screenshot of the annotation panel for Experiment 2 with the four options (from left to right): very useful, useful, okay and spam. The *useful* option is selected.

Annotation Generator

The *value* annotation definition has an associated annotation generator like the three annotation definitions from Experiment 1. Though compared to the first experiment, there is no initial training data available, so the model will be trained

from the very beginning using the active learning integration of ActiveAnno. For the uncertainty sampling, the predictions with the least amount of associated confidence or probability will be sampled.

4.4.2 Experimental Design

The initial plan was to compare two conditions, the first one using randomly sampled training data to train multiple model versions through incremental learning and the second one using active learning with uncertainty sampling to train multiple model versions. During the experiment, surprising results for the first two conditions meant that three additional conditions were investigated to gain more insights into these results.

Condition 1: Random Sampling with Four Tag Set Options

The first condition used randomly sampled documents in an incremental way. For every 50 to 100 annotated documents, a new version of the model was trained. The first three versions used 50 documents per version, the last three versions used 100, resulting in six versions of the model. It uses the four described tag set options. In total, 450 documents were annotated and used for training.

Condition 2: Active Learning with Uncertainty Sampling and Four Tag Set Options

The second condition starts with 50 random samples, but then switches to sampling documents to annotate by least confidence based on the predictions of the first model trained on those first random samples. The model gets updated every 50 annotated documents and the updated model is used to re-predict all the remaining documents. This updates the predictions and their associated probabilities for those documents. The updated probabilities are then used for the uncertainty sampling by least confidence again. The goal here is to always annotate those documents where the model is most uncertain to give the model the maximum amount of new information to predict the next documents. It also uses the four tag set options and annotated 450 documents for training.

Condition 3: Active Learning with Uncertainty Sampling, Three Tag Set Options and Smaller Increments

As the results of the second condition were counterintuitive, it was tweaked and repeated to make sure the results were not due to some of the parameter choices. Therefore, the initial random samples were reduced to 15 and a new model was trained every 10 newly annotated documents for 20 iterations, resulting in 205 training samples. As the number of documents annotated for the *very useful* tag set option was very low, it was combined together with the *useful* option to reduce the complexity of the task.

Condition 4: Random Sampling with Three Tag Set Options and Pre-Trained Vectors

Condition 3 yielded the same results as Condition 2, so an additional change was made. While Condition 1 - 3 used only the supervised classifier of fastText, Condition 4 combines it with pre-trained word embeddings or *pre-trained vectors*. For this, all documents used for Experiment 1 were combined into a single text file and fed into the `skipgram` feature of fastText[5]. This creates a `.vec` file where every word from the input file has an associated vector. This file can be passed to a supervised model with the `pretrainedVectors` parameter[6]. By using the existing documents from Experiment 1, 11,288 context-specific word representations were learned upfront. The hypothesis is that by combining unsupervised and supervised learning, the model does not need to learn the word representations *and* the correct labels but instead will have low confidence values specifically because of unknown labels, not unknown words.

To create this model, the identical 450 annotated documents from Condition 1 were used. Similar to Condition 3, the *very useful* and *useful* tag set options were merged because the amount of documents for *very useful* were too low. Because the annotations from Condition 1 were reused to train this model, only one (final) model version exists for this condition.

Condition 5: Active Learning with Uncertainty Sampling, Three Tag Set Options and Pre-Trained Vectors

Condition 5 uses the identical pre-trained vectors as Condition 4 and also uses the reduced number of tag set options. The sampling approach is the same as Condition 2, with 50 initial random samples and 50 samples per iteration sorted by least confidence based on the current model until 450 annotated documents are reached. For this condition, 450 new documents were annotated, as the uncertainty sampling will be influenced by the pre-trained vectors.

4.4.3 Execution

The conditions were done in the order as defined by their number. In general, every iteration of a condition started with annotating `n` (10 - 100) documents, depending on the condition. After that, the ActiveAnno API (see Subsection 3.5.1) was used to trigger an update of the annotation generator, which trained a new version of a supervised fastText model in the external machine learning service. After that, the annotation value was (re-)generated for every document which was not already annotated by the annotator, also using the ActiveAnno API. Then, the annotate UI was refreshed using the refresh button, which updated the documents in the UI based on the newest sampling. For the active learning conditions, this ensured that always the lowest confidence samples for the newest version of the model were annotated.

[5] `https://fasttext.cc/docs/en/unsupervised-tutorial.html#training-word-vectors`
[6] `https://fasttext.cc/docs/en/options.html`

The time to update the model and to (re-)predict the documents took between 2 and 6 minutes, depending on the condition and version. This included querying the documents for training, the HTTP communication between the two services, the time to load the new fastText model into memory, the actual training, the prediction of 2,500 documents for each condition, and the storing of the results in the database of ActiveAnno. Usually, this could be done in the background without the annotator noticing, with newer generated annotation values being available once the process finished. But due to the structure of the experiment, the annotation process has to wait for the training and prediction to finish. It was not a goal to minimize training and prediction time beyond a certain point for this experiment, so other machine learning implementations and hardware resources for ActiveAnno and the machine learning component might decrease this duration significantly.

For evaluating all the conditions, an additional 600 randomly sampled documents were annotated. These documents were sent to a test API of the external machine learning service, which uses fastTexts existing testing capabilities, the scikit-learn functionality `classification_report`[7] as well as manual calculation of average, median, minimum and maximum confidence values. Finally, the predictions were grouped together based on their confidence values into buckets, similar to a histogram, in steps of 0.1. Through this, it can be analyzed how high the accuracy of the model would be when setting a confidence threshold (e.g. 0.8) for fully automatic annotation.

4.4.4 Results

The initial two conditions C1 and C2 had counterintuitive results: As seen in Figure 4.4, the active learning approach yields dramatically worse results than the random sampling approach with only 59% accuracy compared to 71.33% accuracy for the random sampling model. C3 confirmed the same with 54.5% accuracy. Looking deeper into the data, Figure 4.5 shows the F1 scores per label option for the four models with 450 training samples, revealing that C2 has an F1 score of 0.16 for the *spam* (`SPAM`) option. Therefore, the misclassification of *spam* documents is the major factor in the low accuracy of C2.

At this point, it was required to treat the machine learning component no longer as a blackbox to find out the reasoning for this. As fastText uses vector representations of words internally in the supervised model, it is possible to count the number of known words for each version. A known word is one which fastText has seen during training, and has therefore created a vector representation for. Figure 4.6 shows the new number of known or learned words for each version of the models for C1 / C4, C2 and C5. The number of learned words are normalized over the number of new documents annotated per version, as C1 had some versions with 50 and some with 100 new documents. The figure shows that the uncertainty sampling of C2 is heavily biased towards documents with the most number of

[7] `https://scikit-learn.org/stable/modules/model_evaluation.html#classification-report`

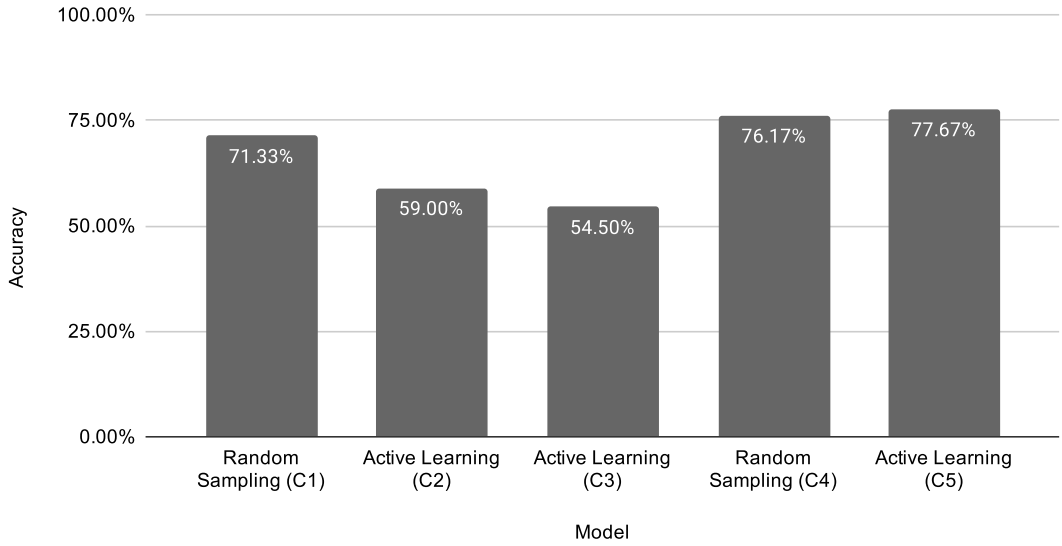


Figure 4.4: Accuracy per model for 600 test samples and the final version of each model.

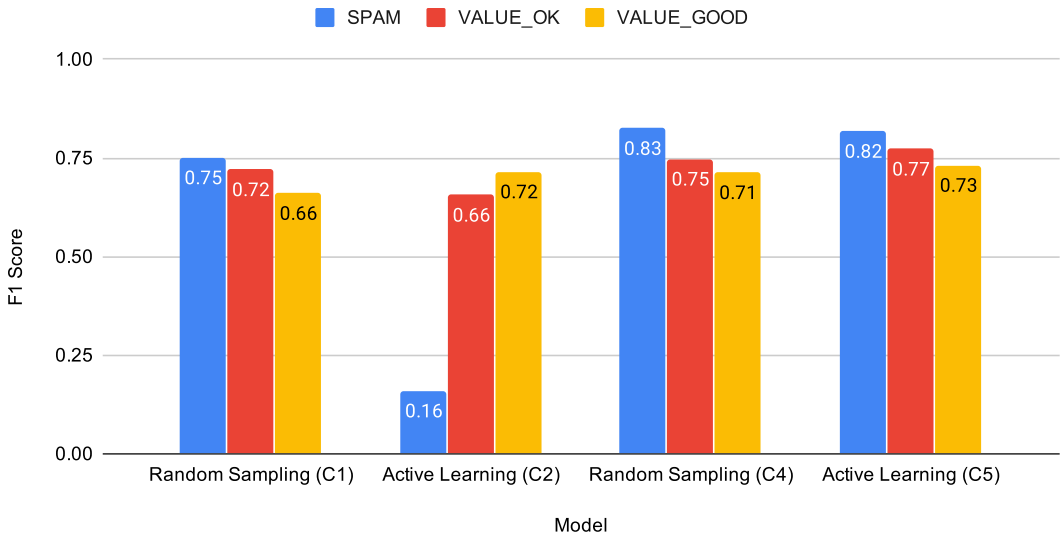


Figure 4.5: F1 score per label option and final version of models C1, C2, C4 and C5. `VALUE_HIGH` was excluded for C1 and C2, because only 2 test samples have that value.

unknown words, as C2 learns a lot more new words for the first three iterations. Comparing that to C1 / C4, there is no such trend as sampling is randomized.

Almost all annotated documents used as training data with a lot of words separated by whitespace characters were annotated as *useful* (`VALUE_GOOD`), while *spam* documents often contain only a single character or a single long token of random

characters. The example comments from Table 4.11 are a good representation for this. Therefore, it appears that low confidence values from fastText predictions can express uncertainty about the label of a document, but also the lack of "understanding" the words of a document.

To reduce this effect, the pre-trained vectors feature of fastText was introduced for C4 and C5. This was successful, as Figure 4.6 shows no priority for learning new words for C5, as the baseline of 11,288 context-specific words is significantly higher than the number of newly learned words for C2, which is 2194.

As seen in Figure 4.5, C5 does not have the same bad F1 score for the `SPAM` label option as C2, but the accuracy score as shown in 4.4 is also only marginally better than for C4. While active learning is no longer worse than the random sampling when using pre-trained vectors, it still did not increase in a significant way. Additionally, Figure 4.7 shows that the initial accuracy for only 50 training samples for version 1 of C5 is already at 74.83%, showing that the pre-trained vectors combined with only 50 random samples are already as powerful as 450 random samples or even 50 random samples combined with 400 active learning samples. When analyzing the average and median confidence per version, the same trend holds true: C5 already has a high average confidence of 0.86 for the first version with 0.91 for the last version, while the first version of C1 has an average confidence of 0.31, the last version 0.763.

4.4.5 Discussion

The initial goal of this thesis was to treat the machine learning part as a blackbox, which worked well for Experiment 1. In the case of Experiment 2, it shows that this is not be possible and that the specifics of fastText need to be taken into account to understand the results. The results show that a supervised fastText model with no preexisting word vectors needs to learn those vectors before it can learn the actual classification task properly. When no pre-trained vectors were available, the sampling was biased towards documents with large numbers of unknown words. This coincides with the analysis of Varghese et al. [24], which concluded that active learning can introduce sampling bias and therefore perform worse.

The results of C5 show that including the unsupervised approach of pre-trained vectors removed that sampling bias. This supports the results of Zhu et al. [27], which combined a semi-supervised approach with active learning. Adding unsupervised learning means that the whole dataset is included in the sampling process, not only the few initial samples queried by the active learning process itself. Therefore, the influence of those few samples is lower, reducing bias.

The results also show that using the pre-trained vectors is very powerful, achieving high accuracy values of 74.83% with only 50 random samples. This power might be the reason that C5 cannot achieve significantly better results than C4 for a scope of 450 training samples, as it appeared that low confidence documents were mostly outliers, which does not significantly help increase the accuracy for a randomly sampled test set. Therefore, other querying strategies for active learning should be investigated in the future in the context of ActiveAnno, to try to prevent a

sampling bias as well as sampling outliers.

Threats to Internal Validity

The fact that even the best performing active learning model of C5 was not significantly better compared to the random sampling approach of C4 might be due to the setup of the experiment. It is possible that the benefits of active learning would have been more significant with a higher number of training samples than 450. The influence of the size of documents per iteration was already compared between C2 and C3 with 50 and 10 documents respectively, which did not show a significant difference. It is also possible that the annotation definition used on the dataset was a bad choice and that for other annotation definitions, for example the sentiment, differences would have been more significant, though it is not obvious why that would be the case. Future work could test this hypothesis by changing the used annotation definition between conditions.

Threats to External Validity

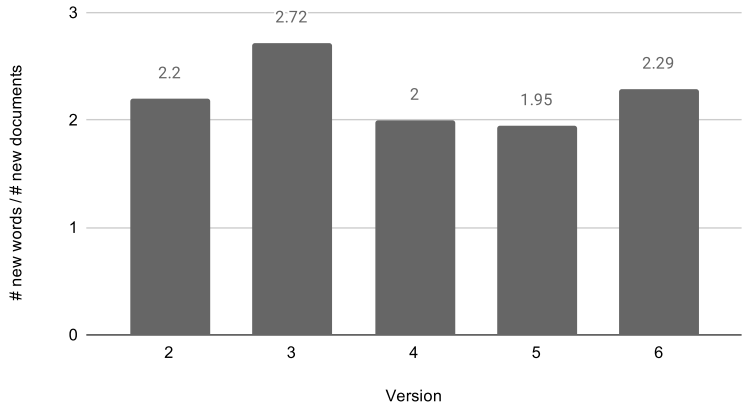
As seen in the initially surprising results of comparing C1 and C2, it is not always possible to treat the machine learning component as a blackbox. Especially since ActiveAnno does not provide inbuilt machine learning, but instead allows anyone to connect ActiveAnno to their own machine learning implementation, every implementation needs to be tested on how it behaves in an active learning setup. This means that the specific results of this experiment do not necessarily generalize to other machine learning approaches. Still, the experiment demonstrates how it is possible to test a machine learning implementation connected to ActiveAnno for active learning and shows the general process of active learning in practice.

Future work for active learning in ActiveAnno can provide support for different active learning querying strategies, for example to sample the most representative documents first, which might work better for certain machine learning implementations. Future experiments could compare different open-source machine learning solutions to analyze which work best for common active learning use cases in ActiveAnno, such that users can have more insights about how their approach might perform in practice.

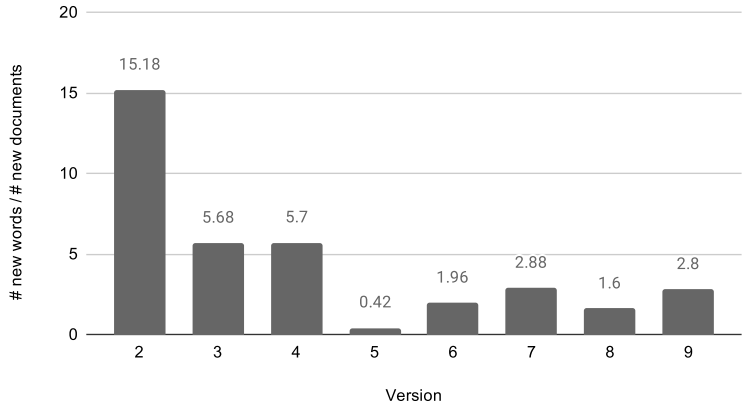
4.4.6 Threshold-based Automation

The benefit of integrating machine learning in the process of annotation is the capability to use the machine learning models for automation. While very good models might be used to fully automate a task and replace human annotators, active learning can be combined with automating the annotation for a subset of the documents to have a more efficient process while keeping control over the annotation quality requirements. This subset of documents would be those for which the machine learning model is very confident in its prediction. To analyze this, predictions for the conditions were grouped into buckets in intervals of 0.1 according to their associated probability or confidence values. For example, a prediction

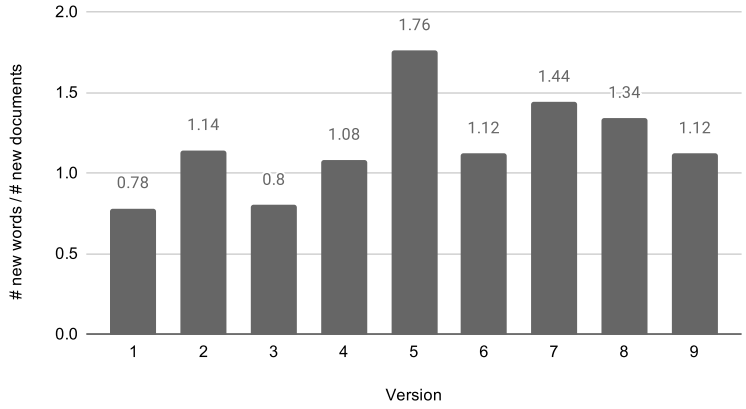
with a probability of 0.94 is placed on the bucket of probabilities from 0.9 to 1.0. Figure 4.8 shows how the confidence values progress for each version of C2 and C5. While C2 achieves to have 23% of its predictions with a confidence value above 0.9, C5 with pre-trained vectors already has 59% of its predictions in that bucket with only 50 random training samples. Figure 4.9 displays the accuracy for the final version of each model for the three buckets with the highest confidence. For all four models, the accuracy for predictions with a confidence above 0.9 is at least 84.1% for C4, up to 88.97% for C2. Therefore, it is possible to use the *generator as annotator* functionality of ActiveAnno and define a finalize condition for the generated annotation of having a confidence value above 0.9. In that case, 23% of documents for C2 with an accuracy of 88.97% would be annotated automatically, or 70% of documents with an accuracy of 85.37% for C5. Every other document would be annotated by a human, but continuously updating the model and predictions would elevate more documents to have a confidence above the defined threshold over time, reducing the actual amount of manual annotations even further. In this scenario, using uncertainty sampling would be the most effective, as documents with an already higher confidence while being below the threshold would eventually reach that threshold and be annotated automatically.



(a) C1 / C4

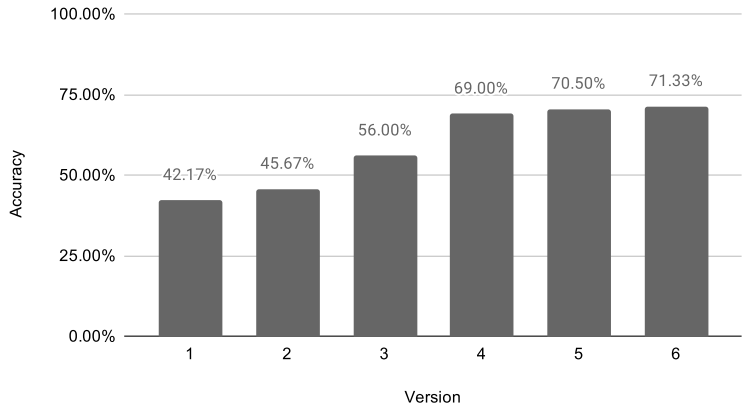


(b) C2

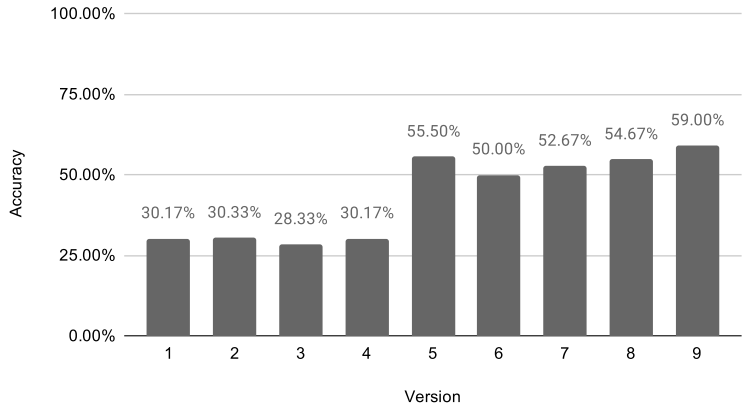


(c) C5

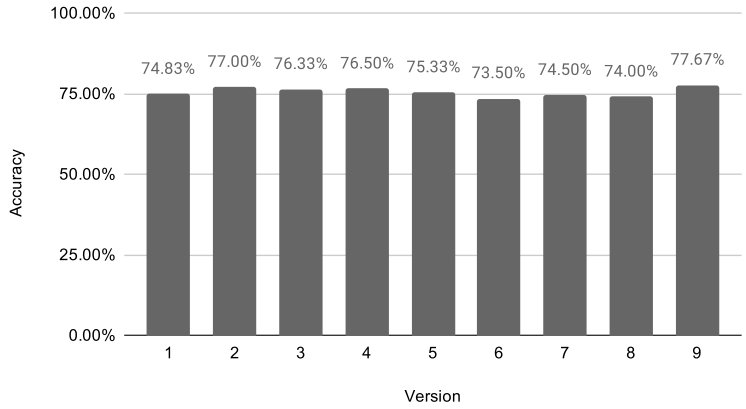
Figure 4.6: Number of newly learned words per new document for each version of the model.



(a) C1

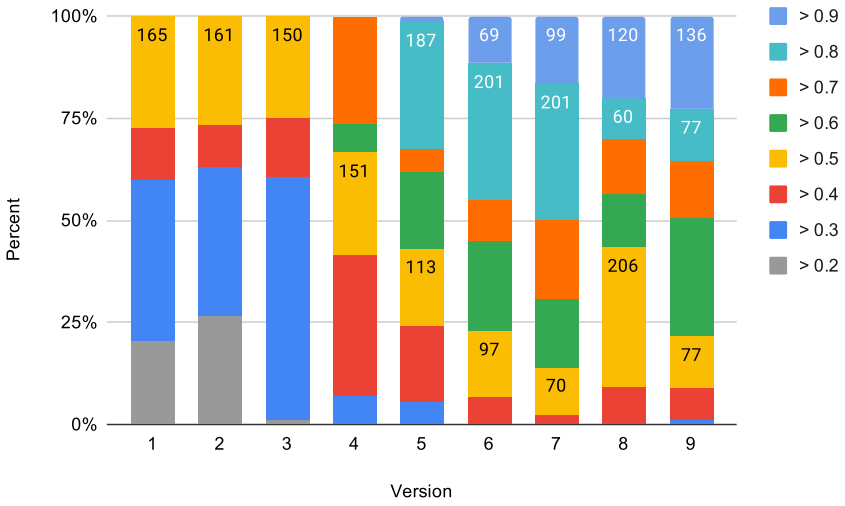


(b) C2

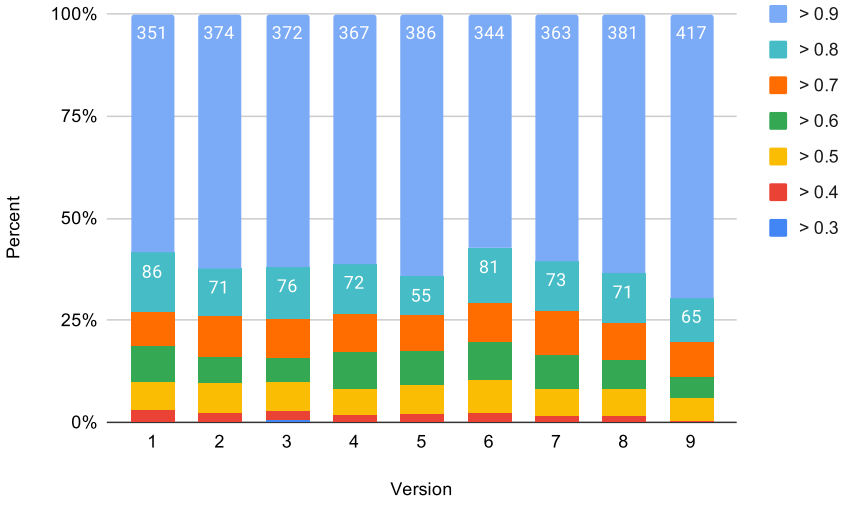


(c) C5

Figure 4.7: Accuracy for every version of the respective model for 600 test samples.



(a) C2



(b) C5

Figure 4.8: Distribution of confidence values for predictions on 600 test samples into buckets from 0.9 to 1.0, 0.8 to 0.9, 0.7 to 0.8 and so on.

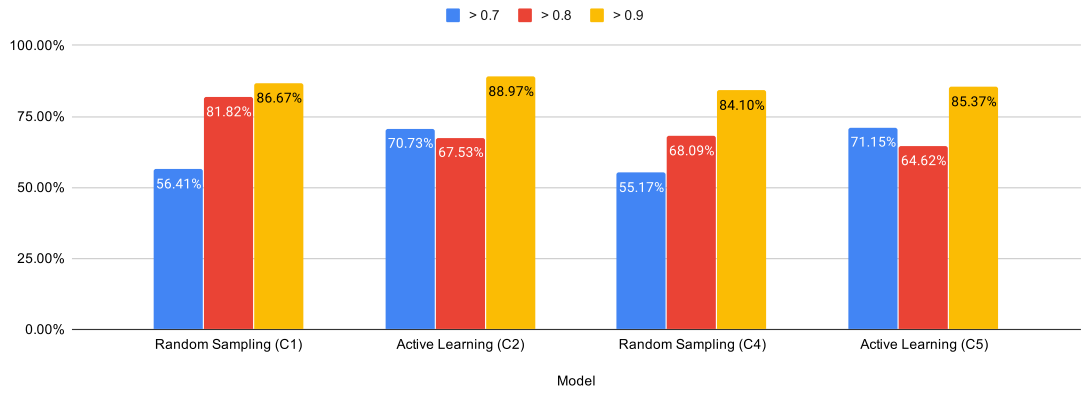


Figure 4.9: Accuracy of the final version of models C1, C2, C4 and C5 on 600 test samples for high confidence buckets above a confidence value of 0.7.

Chapter 5

Conclusion and Future Work

To conclude this thesis, the answers to the two research questions are summarized and ActiveAnno is evaluated on the goals defined in Chapter 3. In addition, potential future work for both research and the developed tool are discussed.

5.1 Research

The two research questions from Section 1.2 were investigated by two experiments in an industry setting, using real-world use cases and data.

5.1.1 Research Question 1

Research Question 1 was answered by Experiment 1: Pre-annotations for document-level annotations increased the efficiency by about 28% while not having a significant effect on annotator accuracy or inter-annotator agreement. Comparing this to the previous work on the effects of pre-annotations (Section 2.4), which were all done on span-level annotations, the increase in efficiency generally coincides with their results. The impact of pre-annotations on the quality of annotations and inter-annotator agreement appears to depend on the actual task and method used to generate the pre-annotations. For potentially complex tasks like named entity recognition in long documents, pre-annotations can help to increase agreement and quality of annotations. But for this comparably simple task with a fixed set of required document-level annotations as well as well-trained annotators, those pre-annotations did not have an impact. Future work could transfer the experiment setup into a setting in which pre-annotations would be more likely to decrease annotation quality, for example by using a worse machine learning model to generate the pre-annotations and/or by using less qualified annotators.

5.1.2 Research Question 2

The results of Experiment 2 show that active learning using uncertainty sampling can introduce a sampling bias which decreases the model performance. The bias was specific to the fastText approach used for this experiment, but this shows that

the choice of the machine learning approach can have a significant impact. As ActiveAnno supports active learning through connecting an external machine learning component to it, users need to analyze the performance of their specific machine learning solution. In this experiment, the negative effects of the initial setup could be removed through the combination with unsupervised learning, which improved the model performance significantly. Future research should investigate different kinds of machine learning approaches as well as query strategies for active learning, such that a reliable general-purpose text classification component could be developed for ActiveAnno in the future.

5.2 Tool

In Chapter 3, five goals were stated for ActiveAnno to be a useful general-purpose document annotation tool.

Annotation Quality and Efficiency ActiveAnno supports multiple mechanisms to control for high-quality annotations and an efficient annotation process. Project managers can define the number of required annotators per document and how agreement is calculated among multiple annotators. The two-step annotation process allows for curators to accept or overwrite annotations created by the annotators. The machine learning integration enables pre-annotations for a more efficient annotation process, as shown in Experiment 1 (Section 4.3). Annotation generators can also be treated as annotators. This enables agreement logic between machine learning models and human annotators as an alternative approach to integrate machine learning into the annotation process. This way, it is also possible to partly or fully automate an annotation task (Section 3.5). The ability to update annotation generators and to sort documents by the least confident predictions enables an active learning process that gradually improves the machine learning models, which can then be used as described above.

Future work for better annotation quality is in-app communication between annotators and between annotators and curators. Similar to a machine learning model improving over time by learning new information, the annotators should receive continuous education by informing them about wrong annotations. Annotators should also be able to directly ask curators if they don't know the correct annotation, to prevent wrong annotations from being created. For onboarding of new annotators, a set of example documents with annotations and explanations would be another useful feature.

The machine learning capabilities of ActiveAnno can also be further improved. Currently, only tag set annotation definitions have an associated annotation generator. Other types of annotation, such as numbers and texts, should be able to use the machine learning capabilities as well. The active learning process should also support more query strategies which might result in better performance, as discussed in Subsection 5.1.2.

Flexibility and Configurability The project configuration of ActiveAnno features a wide range of configuration options. Most importantly, the annotation schema with its annotation definitions is highly configurable. Seven different types of annotation definitions, for example tag sets, numbers and open texts, are supported and can be combined at will. A flexible user management allows to control which user is assigned which role for a project. This makes ActiveAnno a flexible tool, adaptable to different kinds of annotation processes.

Currently, some project configuration options are missing from the user interface. Future work shall make all those accessible from the UI. Especially the layout of the annotation view should be fully customizable with different layouts for different screen sizes. To support even more use cases, span annotations should also be supported in the future. While many tools with span-level annotations already exist, adding support for them in ActiveAnno allows it to be used in scenarios where both document-level and span-level annotations are required. This also enables hybrid annotations, where the annotator can decide to annotate for the whole document or a specific span. To support span-level annotations, the annotation layout needs to be more flexible to better support the display of long documents.

Functionality and Usability ActiveAnno provides a modern, responsive user interface and the annotators from Experiment 1 report a positive user experience (Subsection 4.3.4). The web application supports annotation, curation, management of annotation definitions and projects, as well as the analysis of results with statistics about annotation duration, accuracy and inter-annotator agreement.

In the future, a search function can allow easier access to specific documents and an administration panel can give access to central user management and the ability to configure application-wide settings directly from the UI. Additional usability improvements include a back button for annotators to get to the previous document and revise an annotation, a progress bar for annotators and highlighting differences between annotation results in the curation view.

Interoperability and Installability To integrate with existing software, ActiveAnno provides multiple interfaces. A JSON API enables importing documents and annotations, exporting annotation results and controlling the annotation generators (Subsections 3.3.2 and 3.5.1). Additionally, webhooks can be configured to push finished annotation results to external services. Finally, the machine learning integration as described in Section 3.5 allows for external machine learning components to be integrated through HTTP communication as well. Through Docker, ActiveAnno can be deployed easily to interact with those external services.

While interoperability is helpful for use cases where existing software will be connected, future work for ActiveAnno should also support a standalone mode, where user authentication and machine learning are directly built into the tool. This would make it easier to use all the features of ActiveAnno in cases where a software integration is not wanted.

Extensibility and Open Source While some future extensions of ActiveAnno can already be anticipated, many won't be. Therefore, ActiveAnno was designed with extensibility in mind. Through broad usage of polymorphism in the backend, new types of annotation definitions, layout elements, or annotation generators are easily achievable without changing existing code. To get feedback and input about the tool, ActiveAnno is open-source and open for contributions of any kind under `https://github.com/MaxMello/ActiveAnno`. In conclusion, ActiveAnno and this thesis show how research and industry can collaborate in an open-source context to create shared insights and a useful general-purpose tool for all participants.

Bibliography

[1] Samuel Bayer. MITRE Annotation Toolkit (MAT). http://mat-annotation.sourceforge.net/, 2015.

[2] Juan Miguel Cejuela, Peter McQuilton, Laura Ponting, Steven J. Marygold, Raymund Stefancsik, Gillian H. Millburn, Burkhard Rost, and the FlyBase Consortium. tagtog: interactive and text-mining-assisted annotation of gene mentions in PLOS full-text articles. *Database*, 2014, 04 2014. ISSN 1758-0463. doi: 10.1093/database/bau033. URL https://doi.org/10.1093/database/bau033. bau033.

[3] David Cohn. *Active Learning*, pages 10–14. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_6. URL https://doi.org/10.1007/978-0-387-30164-8_6.

[4] Richard Eckart de Castilho, Chris Biemann, Iryna Gurevych, and Seid Muhie Yimam. Webanno: a flexible, web-based annotation tool for clarin. In *Proceedings of the CLARIN Annual Conference (CAC) 2014*, page online, Utrecht, Netherlands, October 2014. CLARIN ERIC. Extended abstract.

[5] Karën Fort and Benoît Sagot. Influence of pre-annotation on POS-tagged corpus development. In *Proceedings of the Fourth Linguistic Annotation Workshop*, pages 56–63, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W10-1807.

[6] Glenn T Gobbel, Jennifer Garvin, Ruth Reeves, Robert M Cronin, Julia Heavirland, Jenifer Williams, Allison Weaver, Shrimalini Jayaramaraja, Dario Giuse, Theodore Speroff, Steven H Brown, Hua Xu, and Michael E Matheny. Assisted annotation of medical free text using RapTAT. *Journal of the American Medical Informatics Association*, 21(5):833–841, 01 2014. ISSN 1067-5027. doi: 10.1136/amiajnl-2013-002255. URL https://doi.org/10.1136/amiajnl-2013-002255.

[7] Ralph Grishman and Beth Sundheim. Message understanding conference- 6: A brief history. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996. URL https://www.aclweb.org/anthology/C96-1079.

[8] George Hripcsak and Adam S. Rothschild. Agreement, the f-measure, and reliability in information retrieval. *Journal of the American Medical Informatics Association*, 12(3):296 – 298, 2005. ISSN 1067-5027. doi: https://doi.org/10.1197/jamia.M1733. URL http://www.sciencedirect.com/science/article/pii/S1067502705000253.

[9] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics, April 2017.

[10] Todd Lingren, Louise Deleger, Katalin Molnar, Haijun Zhai, Jareen Meinzen-Derr, Megan Kaiser, Laura Stoutenborough, Qi Li, and Imre Solti. Evaluating the impact of pre-annotation on annotation speed and potential bias: natural language processing gold standard development for clinical named entity recognition in clinical trial announcements. *Journal of the American Medical Informatics Association*, 21(3):406–413, 09 2013. ISSN 1067-5027. doi: 10.1136/amiajnl-2013-001837. URL https://doi.org/10.1136/amiajnl-2013-001837.

[11] Jinghui Lu and Brian MacNamee. Investigating the effectiveness of representations based on pretrained transformer-based language models in active learning for labelling text datasets, 2020.

[12] Marcin Michał Mirończuk and Jarosław Protasiewicz. A recent overview of the state-of-the-art elements of text classification. *Expert Systems with Applications*, 106:36 – 54, 2018. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2018.03.058. URL http://www.sciencedirect.com/science/article/pii/S095741741830215X.

[13] Hiroki Nakayama, Takahiro Kubo, Junya Kamura, Yasufumi Taniguchi, and Xu Liang. doccano: Text annotation tool for human, 2018. URL https://github.com/doccano/doccano. Software available from https://github.com/doccano/doccano.

[14] Kimberly A. Neuendorf and Anup Kumar. *Content Analysis*, pages 1–10. American Cancer Society, 2016. ISBN 9781118541555. doi: 10.1002/9781118541555.wbiepc065. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118541555.wbiepc065.

[15] Mariana Neves and Jurica Ševa. An extensive review of tools for manual annotation of documents. *Briefings in Bioinformatics*, 12 2019. ISSN 1477-4054. doi: 10.1093/bib/bbz130. URL https://doi.org/10.1093/bib/bbz130. bbz130.

[16] Ines Rehbein, Josef Ruppenhofer, and Caroline Sporleder. Assessing the benefits of partial automatic pre-labeling for frame-semantic annotation. In *Proceedings of the Third Linguistic Annotation Workshop (LAW III)*, pages 19–26, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W09-3003.

[17] David Salgado, Martin Krallinger, Marc Depaule, Elodie Drula, Ashish V. Tendulkar, Florian Leitner, Alfonso Valencia, and Christophe Marcelle. MyMiner: a web application for computer-assisted biocuration and text annotation. *Bioinformatics*, 28(17):2285–2287, 07 2012. ISSN 1367-4803. doi: 10.1093/bioinformatics/bts435. URL https://doi.org/10.1093/bioinformatics/bts435.

[18] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[19] Burr Settles. From theories to queries: Active learning in practice. In Isabelle Guyon, Gavin Cawley, Gideon Dror, Vincent Lemaire, and Alexander Statnikov, editors, *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, volume 16 of *Proceedings of Machine Learning Research*, pages 1–18, Sardinia, Italy, 16 May 2011. PMLR. URL http://proceedings.mlr.press/v16/settles11a.html.

[20] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 287–294, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130417. URL https://doi.org/10.1145/130385.130417.

[21] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing Management*, 45(4):427 – 437, 2009. ISSN 0306-4573. doi: https://doi.org/10.1016/j.ipm.2009.03.002. URL http://www.sciencedirect.com/science/article/pii/S0306457309000259.

[22] Brett R. South, Danielle Mowery, Ying Suo, Jianwei Leng, Óscar Ferrández, Stephane M. Meystre, and Wendy W. Chapman. Evaluating the effects of machine pre-annotation and an interactive annotation interface on manual de-identification of clinical text. *Journal of Biomedical Informatics*, 50:162 – 172, 2014. ISSN 1532-0464. doi: https://doi.org/10.1016/j.jbi.2014.05.002. URL http://www.sciencedirect.com/science/article/pii/S1532046414001191. Special Issue on Informatics Methods in Medical Privacy.

[23] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, March 2002. ISSN 1532-4435. doi: 10.1162/153244302760185243. URL https://doi.org/10.1162/153244302760185243.

[24] Arun Varghese, Tao Hong, Chelsea Hunter, George Agyeman-Badu, and Michelle Cawley. Active learning in automated text classification: a case study exploring bias in predicted model performance metrics. *Environment Systems and Decisions*, 01 2019. doi: 10.1007/s10669-019-09717-3.

[25] Seid Muhie Yimam, Iryna Gurevych, Richard Eckart de Castilho, and Chris Biemann. WebAnno: A flexible, web-based and visually supported system for distributed annotations. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 1–6, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/P13-4001`.

[26] Seid Muhie Yimam, Chris Biemann, Richard Eckart de Castilho, and Iryna Gurevych. Automatic annotation suggestions and custom annotation layers in WebAnno. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 91–96, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-5016. URL `https://www.aclweb.org/anthology/P14-5016`.

[27] Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML 2003 workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, pages 58–65, 2003.

Erklärung der Urheberschaft

Hiermit versichere ich an Eides statt, dass ich die vorliegende Masterthesis im Studiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum Unterschrift

Erklärung zur Veröffentlichung

Ich stimme der Einstellung der Masterthesis in die Bibliothek des Fachbereichs Informatik zu.

Ort, Datum Unterschrift