ActiveAnno: General-Purpose Document-Level Annotation Tool with Active Learning Integration

Max Wiechmann, Seid Muhie Yimam, Chris Biemann
Language Technology group, Universität Hamburg, Germany
mail@maxwiechmann.de
{yimam,biemann}@informatik.uni-hamburg.de

Abstract

ACTIVEANNO is a novel annotation tool focused on document-level annotation tasks developed both for industry and research settings. It is designed to be a general-purpose tool with a wide variety of use cases. It features a modern and responsive web UI for creating annotation projects, conducting annotations, adjudicating disagreements, and analyzing annotation results. ACTIVEANNO embeds a highly configurable and interactive user interface. The tool also integrates a RESTful API that enables integration into other software systems, including an API for machine learning integration. ACTIVEANNO is built with extensible design and easy deployment in mind, all to enable users to perform annotation tasks with high efficiency and high-quality annotation results.

1 Introduction

Lots of tasks in industry and research require the manual annotation of a potentially large number of documents, for example in content analysis research or supervised machine learning. Existing tools, such as WebAnno (Yimam et al., 2013), allow for span-level annotation tasks like NER and POS tagging. As Neves and Ševa (2019) point out in their review of annotation tools, document-level annotation is a feature missing in most existing tools. ACTIVEANNO is a novel tool created to fill this void. It was created based on five central design goals: 1) Creating annotations of high quality in an efficient manner (**quality**). 2) Support a broad range of use cases without any additional programming effort (**configurability**). 3) provide a good user experience through a modern, responsive web application to be more attractive than specialized prototypes of annotation software (**responsiveness**). 4) Publicly available to extend for future use cases (**open source**). 5) Provide APIs for easy integration into existing software systems and easy deployment to minimize the upfront effort for using ACTIVEANNO (**extensibility**). Through this approach, we would like to put forward ACTIVEANNO as a candidate for the default option for document-level annotation in industry and research.

2 Related Work

Neves and Ševa (2019) conducted an extensive review of 78 annotation tools in total, comparing 15 which met their minimum criteria. Five of those tools support document-level annotations: MAT, MyMiner, tagtog, prodigy, and LightTag. MAT[1] (Bayer, 2015) is designed for what they call the "tag-a-little, learn-a-little (TALLAL) loop" to incrementally build up an annotation corpus, but it is only a research prototype and is not ready to be used in production. MyMiner (Salgado et al., 2012) is an online-only tool without a login or user system. Its main purpose is to classify scientific documents in a biomedical context, which is limited in scope and configuration options and not suitable as a general-purpose tool. The tools tagtog[2] (Cejuela et al., 2014), prodigy[3], and LightTag [4] are all feature-rich and support some form of machine learning integration, but are commercial with either no free version or a free version with limited functionality. This makes these tools less accessible for projects with limited monetary resources.

According to Neves and Ševa (2019), WebAnno is the best tool fitting their criteria, however, it does not support document-level annotations. WebAnno is a notable example of an annotation tool which is open-source, widely used and feature-rich. We adapt some of the functionalities into our ACTIVEANNO document-level annotation tool.

[1] http://mat-annotation.sourceforge.net/
[2] https://www.tagtog.net/
[3] https://prodi.gy/
[4] https://www.lighttag.io/

Another annotation tool with limited support for document-level annotations is Doccano[5] (Nakayama et al., 2018), 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 labeling, and sequence to sequence tasks.

3 Architecture of ACTIVEANNO

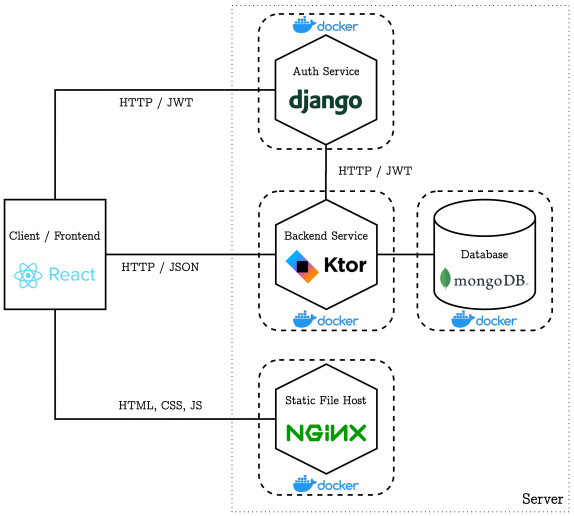

Figure 1: The system architecture of ACTIVEANNO.

ACTIVEANNO is a client-server application as depicted in Figure 1. On the server-side, the main component is a backend service written in Kotlin using the Ktor framework. The backend service is stateless, allowing for horizontal scaling for advanced performance requirements. It is connected to a MongoDB database that is used to store documents and annotation projects. There is also a minimal authentication service written in Python and Django, which provides a simple user management UI and the authentication mechanism of JSON web token (JWT) that is used by the frontend and backend components. The authentication part is extracted into this service such that it can be exchanged for an existing or a more sophisticated user management service, if needed. Finally, the frontend is written in Javascript using React, which is served by an Nginx. All backend components are deployed as Docker containers through existing Docker Compose files. The communication between the client and server is done through HTTP requests via the JSON format.

ACTIVEANNO documents: A *document* is represented as a JSON object with a textual field containing the *document text*, for example the text of a tweet or the abstract of a paper. It can optionally have additional fields with *meta data* that can also be displayed to support the annotation process. Metadata examples include the date and username for tweets, the authors and publication for papers, or the number of stars for an app review. Documents are imported through the UI or API in the JSON format and can be annotated for multiple annotation projects afterward.

Annotation definitions: Annotation projects define an annotation task for one or more *annotation definitions*. An annotation definition describes the required type and form of the annotation that should be created. The typical example is a tag set annotation, where a document is annotated with one or more predefined tags such as *spam/no spam*, the *sentiment* or the *topic* classification of a text. ACTIVEANNO also supports boolean, number, and text annotations with various configuration options such as minimum/maximum values.

Annotations and annotation results: For an annotation project, every document can be annotated for every annotation definition of the annotation project, resulting in one annotation per definition. Together, all annotations for a document from a single annotator form an *annotation result*. Every annotation of the annotation result has a different value structure depending on the type of annotation definition. Additionally, every annotation value can have an associated probability used in the context of automatically generated annotations through the machine learning integration.

Annotation process: 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 4), or they can be imported through the import annotation API (see Section 3.2). After every new annotation result, a 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. The logic is dependent on the project configuration. 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 the document is annotated by three users. One annotator can only annotate every

[5] https://github.com/doccano/doccano

document once. After three annotation results were created, the finalization policy is applied. Depending on the selected policy, the annotation results will either be exported individually, or the annotations are merged based on a majority calculation. If no majority is observed, the document can either be presented to an additional annotator until a majority is reached, or to a curator to decide on the correct annotation result. It is also possible to always require a curator for use cases with very high-quality requirements.

3.1 Web UI

ACTIVEANNO is a modern single-page and responsive web application. It is fully localized, currently supporting English and German. By using a persistent web database, the application state and therefore the created annotations are automatically persisted. A configurable role system allows for the proper authorization of endpoints. The two main roles are *user* and *manager*. A user can be an 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. Though slightly modified, the user roles were inspired by WebAnno's user management.

The web interface is composed of the login page, a landing page and pages for annotation, curation, and management. On the landing page, users can see their areas of responsibility and switch the application language. The manage page is for users with the manager role, allowing them to create and edit projects, annotation definitions, annotation generators, and the analysis of annotation results.

Figure 2 (left) shows the UI (Mobile version) for editing the basic properties of an example project, like name and description. Besides, the other parts of the *manage project* component allow configuring 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. Additionally, the manager of the project can configure the annotation schema, which is composed of the annotation definitions for the project. From this and the *document mapping* step, where the manager defines which part

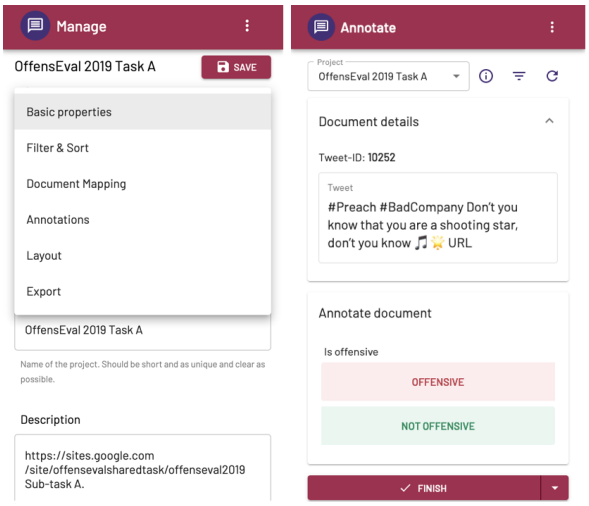

Figure 2: Left: Edit project UI, Right: Annotation UI

(as JSON keys) of the imported documents are relevant for the project, the layout for the annotation task gets generated. The generated layout shows all the metadata, the document text, and the annotation definitions with default input types. Figure 2 (right) shows an example layout for the annotation view (Mobile version). Lastly, the manager can configure how annotation results are able to be exported: either through the REST API, webhooks, or both (see Section 3.2 for the details of the API).

Figure 3 shows the annotation UI for a more complex annotation task, in this case on a desktop layout, with six annotation definitions of different types, including Yes/No annotations, single and multi-select tag set annotations (some displayed as buttons, some as drop-down inputs), a number annotation visualized as a slider, an extensible tag input (where annotators select tags created by other annotators or create their own), and an open text input. After a manager created a project and documents are imported into ACTIVEANNO, the annotators can annotate the documents according to the project set up in the annotate subarea. Depending on how the project is set up, the annotated documents might be checked and possibly overwritten by a curator in the curation subarea. In the curation UI, curators can see all previously created annotation results. Curators have the authority to decide if an annotation created by annotators is correct, to provide a final verdict.

After annotation results are finalized, they can be analyzed by the project managers. In addition to accuracy, the inter-annotator agreement (simple agreement with exact matching), as well as the annotation duration is generated as charts in the UI. There is also a table of individual documents, showing the correctness of every annotator and

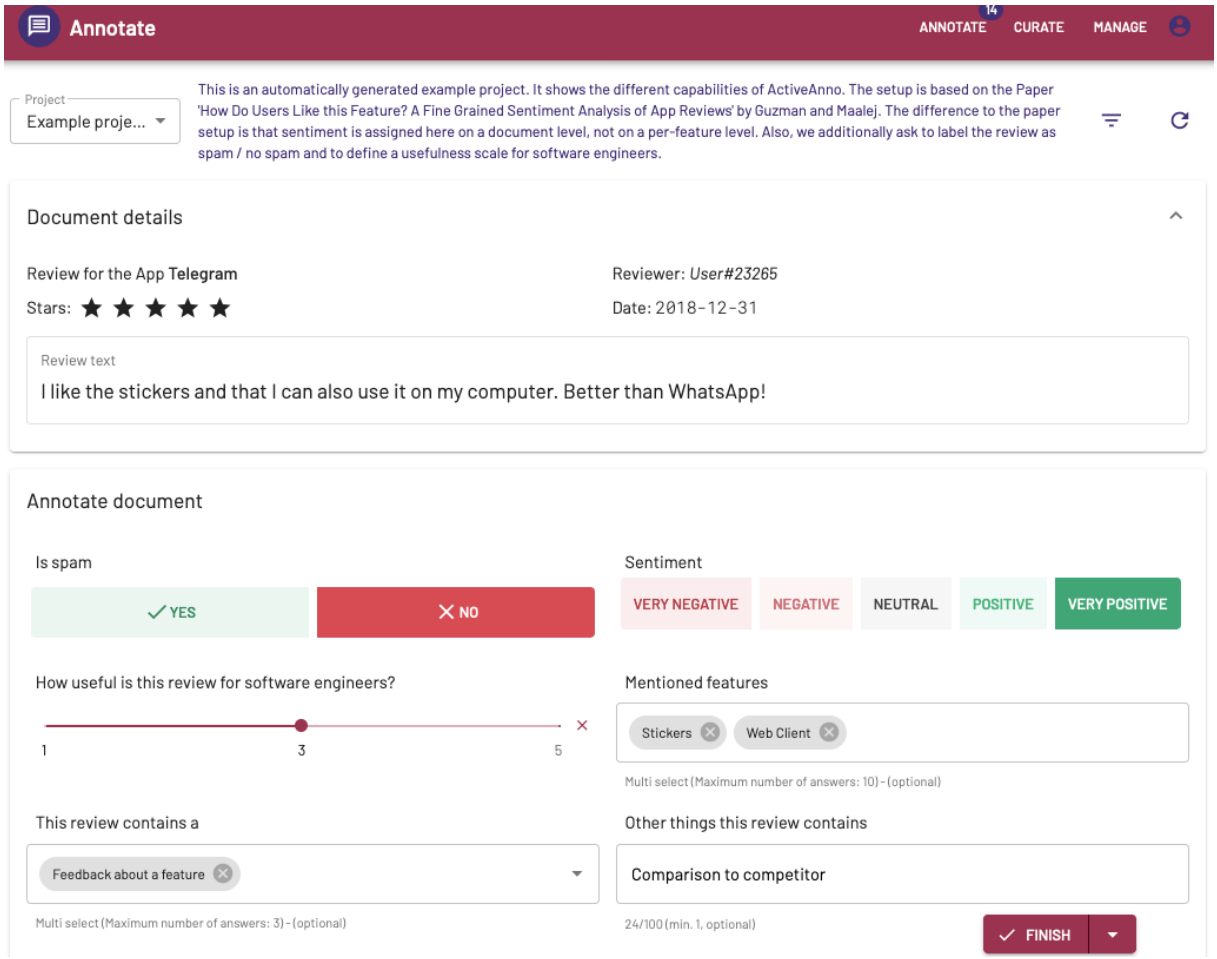

Figure 3: Complex annotation example

their agreements for the document. The UI has filter inputs to restrict analyzed annotation results by *annotator*, *date* range, or more fine-grained filters.

3.2 Import and Export API

ACTIVEANNO provides multiple API endpoints to enable automated document processing and integration into larger software systems. The import document endpoint allows uploading any JSON document or an array of JSON documents. Documents will be assigned a unique ID, stored in the MongoDB, and the IDs will be returned to the caller, so they can be used for future reference, e.g. to export from ACTIVEANNO. The ID is also used to import externally created annotations into ACTIVEANNO for a specific project and document. Imported annotations can be treated like any other created annotation in the agreement process, or as pre-annotations for human annotators.

Through a single export endpoint, annotation results can be exported into other applications for further processing. Through the GET parameters, it can be filtered which annotation results will be exported, for example, based on the timestamp or document IDs. As an alternative to the export API, every project can define a list of webhook URLs, which will be called once a document is finalized.

4 Annotation Generator

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. 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. An annotation generator is defined through an abstract class with a `generateAnnotation` method that every generator needs to implement. Every annotation 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, allowing for chaining or conditional connecting of annotation generators.

Currently, ACTIVEANNO has three inbuilt annotation generator implementations. The first one automatically detects the language of the generator input using the language detection library Lingua[6]. 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 calling 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. The model would have to be wrapped by an HTTP API to comply with the API definition of ACTIVEANNO for this annotation generator. 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. To support this functionality, the concept of an **updatable** annotation generator exists. This kind of generator extends the base annotation generator, but also requires its subclasses to implement an `update` method, 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 are generated, and then it should be updated for every 25 new training samples. An updatable annotation generator is versioned with a version number and an update state, to ensure the version is actually usable to generate new annotations.

Annotation generators can be triggered to generate/re-generate annotations for a project when appropriate, and they can be triggered to update themselves if they are updatable annotation generators and enough new training data is created to allow for a model update.

[6] https://github.com/pemistahl/lingua

4.1 Machine Learning Integration

Once the annotation definitions and annotation generators are created and an external machine learning service is provided in compliance with the API of ACTIVEANNO, the last step is to integrate the machine learning module into ACTIVEANNO. 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 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. Alternatively, it is possible to set the generator results to be treated equally to an annotator where the results will be included in the finalization logic.

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. In this case, if only a subset of documents has 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 to support active learning, where the documents with the lowest confidence values associated with the generated annotations will be preferred (see Section 4.2). We also have an alternative approach for machine learning integration that relies on the import annotation API. Imported annotations can be used instead of internally generated annotations. 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.

4.2 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. 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 a threshold of the annotation generator is reached, triggering the update of the generator will result in the training data being aggregated and sent to the external service. Triggering the *generate annotations* functionality (from the UI or via an API) will result in the newly trained generator model that will create predictions for all remaining documents in the project. Afterwards, 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 for annotation, those with the lowest confidence being presented first.

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 (Bayer, 2015). If combined with enabling pre-annotations, it is also very similar to the annotation process of RapTAT (Gobbel et al., 2014). 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 configuration option of 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 further. To fully automate the process, the *finalize* condition should be set to *always* to accept the annotations automatically without additional conditions.

5 Use Cases

ACTIVEANNO is designed to be applicable in many settings, including industry and research; for small projects on a local machine and large projects with many users on the internet; for use cases with and without machine learning components. We have used ACTIVEANNO in two industry setups, to test its suitability to collect real-world annotation requirements. It was directly deployed on the service cluster of a company, configured via environment variables, and connected to the existing system via its API with some small additions to the existing software system to get documents into ACTIVEANNO. For the machine learning integration, a small Python service wrapping fastText (Bojanowski et al., 2017) models was employed. The data for the experiments were around 250,000 German textual feedback comments obtained from a retail context.

The first experiment was set up to analyze the effects of pre-annotations on annotation quality and efficiency. Three annotation definitions, namely spam/not spam, sentiment, and topic classification were annotated by two annotators employed at the company. One condition was provided without pre-annotations while the other one was provided with pre-annotations for all three annotation definitions. The annotations were created by pre-trained machine learning models that are integrated into ACTIVEANNO through the annotation generator and the machine learning-related APIs. Through the *analyze* functionality of ACTIVEANNO, the inter-annotator agreement (IAA), annotation duration, and the accuracy of annotators as well as the machine learning models are compared inside the tool. The final result showed a 28% faster annotation without any change of annotation accuracy or annotator agreement for the condition with pre-annotations. ACTIVEANNO enabled the research project itself while the annotators reported a positive user experience using the tool. They reported that the tool is easy, fast, and convenient to use.

The second experiment explored the incremental and active learning capabilities of ACTIVEANNO. A new annotation definition about the utility (or information quality) of a comment with three classes (useful, okay, spam) was created as it was a new business requirement to create such annotations for further aggregation and processing. During the experiment, multiple new machine learning models for different conditions were created by annotating new comments inside ACTIVEANNO and triggering the automatic annotation generator updating and re-annotating functionality. By additionally enabling pre-annotations, annotators had reduced effort for selecting the correct annotation option, once the model had an acceptable level of accuracy. Selecting the best performing model based on the experiment conditions then enables and improves the regular annotation process of the company. Once the model performs well enough while being used for pre-annotation, it could then be used to partly automate the process as described by simply editing the project configuration.

Finally, as both experiments were conducted on non-publicly available data, we created another example project based on the OffensEval 2019 shared task (Zampieri et al., 2019), specifically sub-task A. The task is directly integrated as an

example project in ACTIVEANNO, including a machine learning component that can be updated through ACTIVEANNO. Please refer to the demo[7] and video[8] to see this use case in action.

6 Conclusion and Future Work

ACTIVEANNO supports several mechanisms to produce high-quality annotations with an efficient annotation process, like the number of annotators per document, a configurable agreement logic, curators, machine learning integration through annotation generators, pre-annotations, treating annotation generators as annotators, partly or fully automating annotations, and updating annotation generators with incremental or active learning. A fully configurable annotation schema with annotation definition types like tag sets, numbers and texts, a modern and responsive web UI, as well as flexible user management allows ACTIVEANNO to be adaptable to many different kinds of annotation process requirements. The RESTful API and webhooks allow for easy integration with other software components.

Future work planned is to add in-app feedback between curators and annotators for improving annotator performance, adding more in-built annotation generators for other types of annotation definitions, UI improvements (layout editor, better display of very long documents) and span-level annotations as well as hybrid-level annotations which can be defined either on a span or document level.

Most importantly, ACTIVEANNO was designed with extensibility and flexibility in mind. It is available as open-source software under the MIT license.

References

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

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Juan Miguel Cejuela, Peter McQuilton, Laura Ponting, Steven J. Marygold, Raymund Stefancsik, Gillian H. Millburn, Burkhard Rost, and the FlyBase Consortium. 2014. tagtog: interactive and text-mining-assisted annotation of gene mentions in PLOS full-text articles. *Database*, 2014. Bau033.

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. 2014. Assisted annotation of medical free text using RapTAT. *Journal of the American Medical Informatics Association*, 21(5):833–841.

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

Mariana Neves and Jurica Ševa. 2019. An extensive review of tools for manual annotation of documents. *Briefings in Bioinformatics*, 22(1):146–163.

David Salgado, Martin Krallinger, Marc Depaule, Elodie Drula, Ashish V. Tendulkar, Florian Leitner, Alfonso Valencia, and Christophe Marcelle. 2012. MyMiner: a web application for computer-assisted biocuration and text annotation. *Bioinformatics*, 28(17):2285–2287.

Seid Muhie Yimam, Iryna Gurevych, Richard Eckart de Castilho, and Chris Biemann. 2013. 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.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019. SemEval-2019 Task 6: Identifying and categorizing offensive language in social media (OffensEval). In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 75–86, Minneapolis, Minnesota, USA.

[7] Source code, documentation and the link to the demo can be found here: https://github.com/MaxMello/ActiveAnno

[8] Demo Video: https://youtu.be/ryCi4XeReDg

A Supplemental Material

The following supplemental material features various screenshots of ACTIVEANNO in order to display the scope of the project.

A.1 Home Page

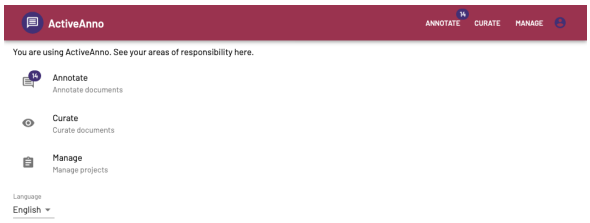

Figure 4: Home page

Figure 4 shows the home page of ACTIVEANNO, including navigation options to all subareas as well as the language switcher.

A.2 Manage UI

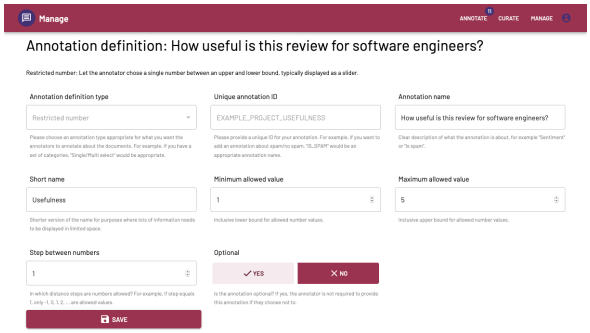

Figure 5: Editing an annotation definition

The manage UI has various parts, such as the ability to edit annotation definitions as depicted in Figure 5, here a number annotation. Another part is creating and editing annotation projects. Figure 6 shows the basic properties of an annotation project, like name, description and user associated to their roles. Figure 7 shows the annotation schema, which is comprised of all the annotation definitions of the project. Finally, Figure 8 shows part of the analysis UI where annotation results can be statistically evaluated. It features graphs of annotation duration, inter-annotator agreement, and accuracy. Not depicted are the filter options to only analyze a subset of all annotation results of the project, for example, based on the annotators or time, and an extensive table showing statistics (agreement, correctness of annotation) for every document analyzed.

A.3 Curation UI

The curation UI, as shown in Figure 9, is an extension of the annotation UI, featuring an additional

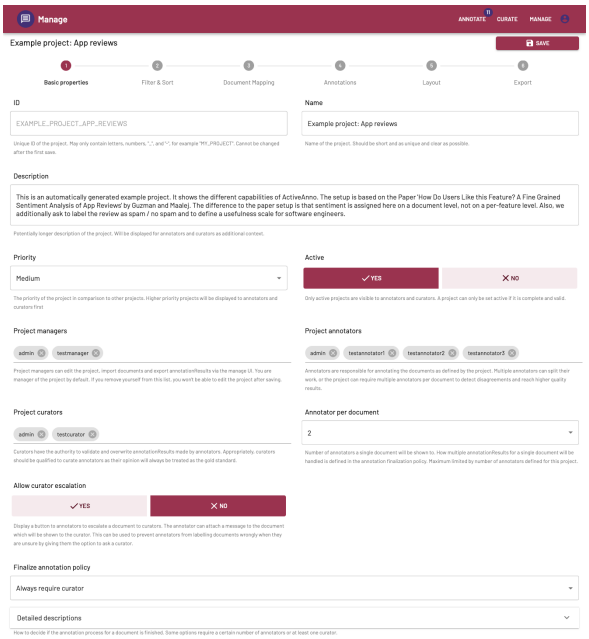

Figure 6: Editing project basic properties

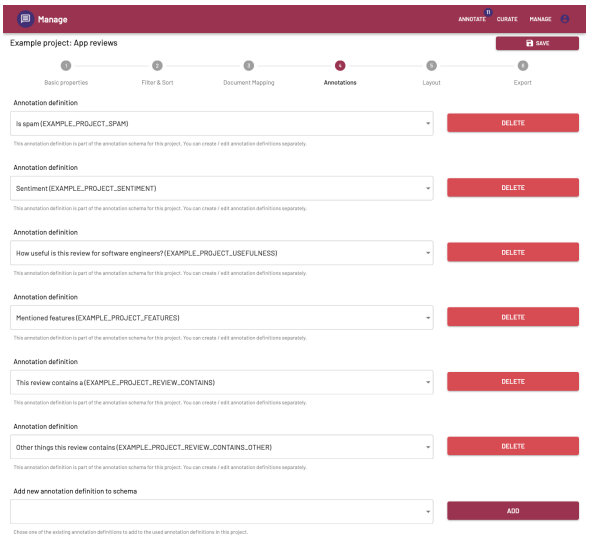

Figure 7: Editing project annotation schema

panel where all annotation results for the document are displayed. The curator can either accept one of those results as correct, copy the result into the annotation area below, or annotate the document themselves in said annotation area.