

AnnoRank Documentation

AnnoRank is a web application tool that has the purpose of collecting annotations given a query and a ranked list of items. The tool has three main functionalities: one that can be used to collect interactions between the user and the ranked list of items, one that can be used to collect graded relevance for an item given the displayed query, and one that can be used to compare two rankings and assess which ranking is more suitable given the query and the assessment's requirements. Moreover, AnnoRank offers the researcher the possibility to view the annotations collected and compare two rankings as well as viewing the corresponding evaluation metrics.

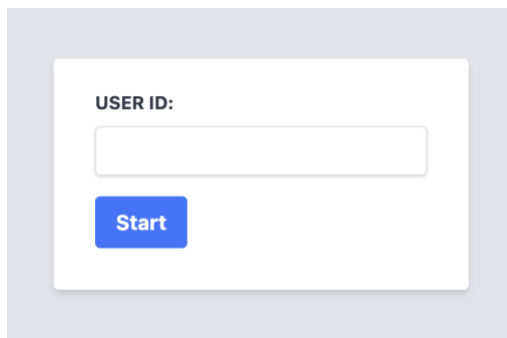
Table of Contents

Table of Contents	1
1. UI Functionalities	2
1.1 Collection of User Data	2
1.2 Display Interaction Annotate UI	3
Integration with BigBrother Logging Tool	4
1.3 Score Annotate UI	5
1.4 Display Ranking Comparison UI	6
2. Displaying a new Dataset	8
3. Configuration File	10
3. Experiment File	13
4. Database	14
5. Additional Support	20
5.1 Ranklib Rankers	20
5.2 Fairness Interventions	23
6. Run the tool	27
7. Ready to run Use Cases	29
7.1 Retail	29
7.2 Recruitment System	31
7.3 Multimodal	33
8. Tutorial	35
Repository Structure	37
References	39

1. UI Functionalities

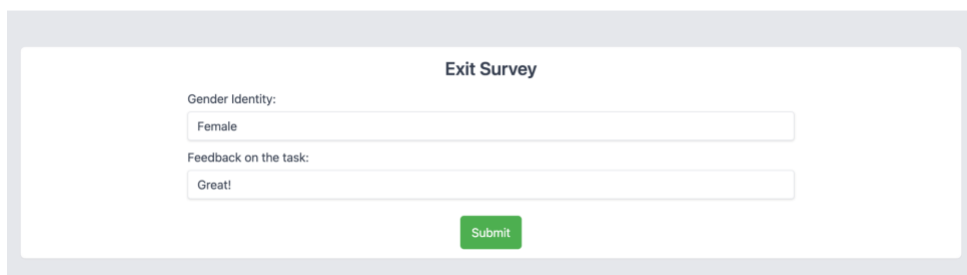
1.1 Collection of User Data

When interacting with the tool the users will be first asked to provide an ID with which they will be registered in the database. The user can interact with the UI by typing in the empty field the ID and then clicking start. All interactions will be collected during the session for each user ID.



A screenshot of a user registration form. It features a white rectangular box with a light gray border. Inside the box, the text "USER ID:" is displayed in a small, bold, black font. Below this text is a white text input field with a thin gray border. At the bottom of the box is a blue rectangular button with the word "Start" written in white, bold, sans-serif font.

At the end of the interaction with the tool, the users can be asked to complete an exit survey in which one can collect feedback or demographic data about the user. An example of an exit survey can be seen below. This can be configured to include other fields. More details about the configuration of the Exit Survey can be found under the section Configuration File. The user can interact with the UI by typing in the empty fields (e.g. feedback field) or by choosing an option from a drop-down (e.g. gender field), then when clicking submit all the collected values are stored in the database for the corresponding user ID.



A screenshot of an "Exit Survey" form. The form is titled "Exit Survey" in a bold, black font at the top center. Below the title, there are two input fields. The first field is labeled "Gender Identity:" and contains the text "Female". The second field is labeled "Feedback on the task:" and contains the text "Great!". At the bottom center of the form is a green rectangular button with the word "Submit" written in white, bold, sans-serif font.

1.2 Display Interaction Annotate UI

Project Manager

Education Background

Degree: bachelors degree

Major: civil engineering or environmental engineering or management engineering

Professional Experience

Role: project manager

Duration: 2 years

Skills Hard

Agile project management, Lean project management, PRINCE2, MS Project, Health and safety regulation, ISO 9001

Skills Soft

Leadership, Risk management, Coaching, Stakeholder management

NAME	EDUCATION	EXPERIENCE	SKILLS		
Manuel Ortega Corral	Degree: Master s Degree	Role: Project manager	Health and safety regulation, ISO 9001, Leadership, Stakeholder management, Lean project management, MS Project, Agile project management, Coaching, Risk management, PRINCE2, Analytics, Incident investigation	View	<input type="checkbox"/>
Isabel Castro Barrera	Degree: Master s Degree	Role: Project manager	MS Project, Risk management, Leadership, Agile project management, Health and safety regulation, Quality assurance, Decision logging	View	<input type="checkbox"/>
Shazia Baig	Degree: Master s Degree	Role: Development project manager	Health and safety regulation, PRINCE2, Risk management, ISO 9001, MS Project, Lean project management, Coaching, Performance monitoring, Analytics	View	<input type="checkbox"/>
Ayesha Younus	Degree: Master s Degree	Role: Engineering project manager	Coaching, ISO 9001, Leadership, PRINCE2, Risk management, Lean project management, Stakeholder management, Performance monitoring, MS Office, Incident investigation, Documentation	View	<input type="checkbox"/>
Carlos Gonzalez Gil	Degree: Bachelor s Degree	Role: Project manager	Agile project management, Coaching, Health and safety regulation, Risk management, ISO 9001, MS Project, Leadership, Earned Value management	View	<input type="checkbox"/>
Cristina Fernandez Medina	Degree: Bachelor s Degree	Role: Project manager	Coaching, MS Project, Risk management, Leadership, Health and safety regulation, Agile project management, ISO 9001, PRINCE2, Lean project management, Online marketing, Statistics, Analytics	View	<input type="checkbox"/>
Fatih Avci	Degree: Master s Degree	Role: Engineering project manager	Health and safety regulation, Leadership, Lean project management, Agile project management, PRINCE2, ISO 9001, MS Project, Stakeholder management, Coaching, PESTEL, Incident investigation, Procurement	View	<input type="checkbox"/>
Aissatou Gaye	Degree: Master s Degree	Role: Development project manager	MS Project, Leadership, ISO 9001, PRINCE2, Agile project management, Risk management, Coaching, Stakeholder management, PESTEL	View	<input type="checkbox"/>
Said Bouchama	Degree: Bachelor s Degree	Role: Project manager	Risk management, Agile project management, Coaching, PRINCE2, Health and safety regulation, ISO 9001, Stakeholder management, Analytics	View	<input type="checkbox"/>
Malika Filali	Degree: Bachelor s Degree	Role: Project manager	Health and safety regulation, Agile project management, MS Project, Stakeholder management, Lean project management, Coaching, Statistics	View	<input type="checkbox"/>
Next					

In the option for displaying a query-ranking pair the UI will show in the top of the page the query, and in the bottom of the page the list of documents. The user can interact with the UI in the following ways:

View Button: the user can expand the table to view more information about the document. When the user clicks on another View Button, the current expand will close. Only, one expand can be open at a time.

NAME	EDUCATION	EXPERIENCE	SKILLS	
Manuel Ortega Corral	Degree: Master s Degree	Role: Project manager	Health and safety regulation, ISO 9001, Leadership, Stakeholder management, Lean project management, MS Project, Agile project management, Coaching, Risk management, PRINCE2, Analytics, Incident investigation	View <input type="checkbox"/>
	Degree: Master s Degree Major: Management engineering Institution: University of Valencia Spain	Role: Project manager Duration: 1 6 years Start Date: Jul 2022 End Date: Feb 2024 Organization: Amazon Spain Services Madrid Spain		
	Degree: Bachelor s Degree Major: Environmental engineering Institution: University of Valencia Spain			

Check Box: the user can select top-k documents that are the most relevant given the displayed query. If the predefined minimum and maximum of selected documents is not met, the UI will display an alert to the user.

Next Button: the user can navigate to the next query using the Next Button.

For each user, the following interactions can be collected:

- **Clicks:** number of clicks for each document on the View Button
- **Timestamps:** for each click the timestamp of the click to open the expand and the timestamp of closing the expand are collected.
- **Top-k Documents:** the list of the selected documents in the order in which the user checked the box

Integration with BigBrother Logging Tool

On top of the above mentioned, AnnoRank is integrated with the BigBrother logging tool. The BigBrother logging tool is one of the application-independent services, with minimal configuration to the web-based user interface [8] (Github source: <https://github.com/hscells/bigbro>).

The tool can record the **mouse clicks** and **movements** of the annotators. The logs are stored in the `./dataset/<data_name>/big_borther.log` file.

To collect *mouse clicks* insert this script into the html body:

```
<script type="text/javascript">
  BigBro.init('**{{session_id}}**', "localhost:1984", ["click",
"onload"]);
</script>
```

To collect *mouse movements* insert this script into the html body:

```
<script type="text/javascript">
  BigBro.init('**{{session_id}}**', "localhost:1984", ["mousemove",
"onload"]);
</script>
```

The **session_id** is the **"user_id"** of the current annotator.

Below is an example of the Big Brother logging tool output for *clicks* when used in the Interaction Annotation UI app:

```
2024-06-03 09:50:26.303 +0000
UTC,**3456**,click,INPUT,shortlist,1,http://localhost:5000/start_ranking/1/
index_ranking/0/view,1229,423,1360,807,
2024-06-03 09:50:26.693 +0000
UTC,**3456**,click,INPUT,shortlist,2,http://localhost:5000/start_ranking/1/
index_ranking/0/view,1233,508,1360,807,
2024-06-03 09:50:27.14 +0000
UTC,**3456**,click,TD,,,http://localhost:5000/start_ranking/1/index_ranking
/0/view,1239,587,1360,807,
```

2024-06-03 09:50:27.477 +0000

UTC,**3456**,click,INPUT,shortlist,3,http://localhost:5000/start_ranking/1/index_ranking/0/view,1236,596,1360,807

Below is an example of the BigBrother logging tool output for the *mousemove* in score

Annotation UI:

2024-06-03 11:19:25.017 +0000

UTC,**23445**,mousemove,H4,,,http://localhost:5003/start_annotate/2/index_annotate/1,692,450,1360,807,

2024-06-03 11:19:25.033 +0000

UTC,**23445**,mousemove,H4,,,http://localhost:5003/start_annotate/2/index_annotate/1,949,368,1360,807,

2024-06-03 11:19:25.05 +0000

UTC,**23445**,mousemove,DIV,,,http://localhost:5003/start_annotate/2/index_annotate/1,1129,336,1360,807,

2024-06-03 11:19:25.069 +0000

UTC,**23445**,mousemove,DIV,,,http://localhost:5003/start_annotate/2/index_annotate/1,1303,322,1360,807,

The logs follow the structure:

<Time of logged event>, <User identifier> , <Method causing the event>, <HTML element>, <Name of HTML element> , <ID of HTML element> , <URL> ,<X position> ,<Y position> , <ScreenWidth> , <ScreenHeight> , <Comment>

1.3 Score Annotate UI

Task Description

Given the domain displayed below, select a score from 1 to 5 for the given candidate. 1 means that the candidate is not a good fit for the given job, while 5 means that he is a perfect fit. We are looking for a candidate for a senior job in the given domain, tha should have a bachelor's degree.

Actuary

CANDIDATE	GENDER	EDUCATION	DEGREE	WORK EXPERIENCE
John	female	4 years	Degree: Bachelor	16 years

Overall Score

1 2 3 4 5

Relevance Label

Next

Navigation Button

In the option for annotating a document by giving it a score, the UI will show in the top of the page the query, and in the bottom of the page document. The user can interact with the UI in the following ways:

- **Score Bar:** the user can give a score to the document by clicking on the numbers in the score bar.
- **Next Button:** the user can navigate to the next query using the Next Button.

For each user, the following interactions can be collected:

- **Score:** the score the user thinks the document should receive given the displayed query and the task description.

1.4 Display Ranking Comparison UI

Ranking Compare Visualise UI

Actuary									
CANDIDATE ID	GENDER	WORK EXPERIENCE	EDUCATION	VIEWS	CANDIDATE ID	GENDER	WORK EXPERIENCE	EDUCATION	VIEWS
ranker_1:preprocessing_1:qualification_fair__qualification_fair					ranker_1:qualification__qualification				
3caee75a-5f28-44d2-819f-82fa085c5883	f	201 months	48 months	12.0	e209730f-e993-4d27-8e9f-730cd3b679d4	m	181 months	126 months	2.0
e209730f-e993-4d27-8e9f-730cd3b679d4	m	181 months	126 months	1.0	258aeeeb-6738-4232-a390-24ba9a0bc552	m	230 months	60 months	1.0
258aeeeb-6738-4232-a390-24ba9a0bc552	m	230 months	60 months	1.0	110b0abf-40af-417e-986b-f41c8d274100	f	128 months	109 months	2.0
110b0abf-40af-417e-986b-f41c8d274100	f	128 months	109 months	1.0	7eec57a5-4400-4ea7-82d6-454ae3296fe7	m	202 months	3 months	0.0
b818ab21-985d-4079-8094-075e8eb3bff4	f	115 months	55 months	0.0	b818ab21-985d-4079-8094-075e8eb3bff4	f	115 months	55 months	0.0
ad856ecb-e6db-4411-bc3d-b64c2511067e	m	85 months	61 months	0.0	ad856ecb-e6db-4411-bc3d-b64c2511067e	m	85 months	61 months	0.0
7eec57a5-4400-4ea7-82d6-454ae3296fe7	m	202 months	3 months	0.0	3a155eb4-fd2a-49b6-88d9-12a9522d4eb9	m	121 months	75 months	0.0
460aaf31-9feb-4d5c-afcc-e5d32e31953d	m	232 months	3 months	0.0	460aaf31-9feb-4d5c-afcc-e5d32e31953d	m	232 months	3 months	0.0
3a155eb4-fd2a-49b6-88d9-12a9522d4eb9	m	121 months	75 months	0.0	3caee75a-5f28-44d2-819f-82fa085c5883	f	201 months	48 months	0.0
Fairness Metrics Selection Parity: 0.53 Exposer Parity: 0.21 Igf: F:1, M:0.76 Utility Metrics MAP: 0.8 NDCG: 0.91					Fairness Metrics Selection Parity: 0.0 Exposer Parity: 0.27 Igf: F:0.11, M:1 Utility Metrics MAP: 0.8 NDCG: 0.68				
Prev					Next				

On top of this, this UI can be used by the researcher to compare rankings produced using different ranking methods or evaluate the impact of a fairness intervention. The UI has the option to display metrics for each ranking, as shown in the bottom of the UI. Moreover, if interaction data is available, the UI can display the average of the interactions collected by the users. The navigation between several ranking comparison can be performed using the **Prev/Next Button**.

Evaluation Metrics supported by AnnoRank

The utility metrics that can be displayed are computed using the Pytreceval library[4] on the displayed ranking. The ready to use fairness metrics that the tool offers are: selection parity, parity of exposer, and IGF (in group fairness). Selection parity measures whether the selection rates between the two groups in the top-k are equal. The parity of exposure measures whether a group's exposure is equal to the exposure of the other group at the top. Lower values mean the ranking offers equal representation/exposure among the groups. IGF is computed as the ratio between the lowest accepted score and the highest rejected score of a group at the top. Higher values mean more in group fairness.

Ranking Compare Annotate UI

Task Description

Select the ranking that is considered to have the best trade-off between usefulness and diversity with respect to the gender, where "f" means female and "m" means male, given the occupation displayed below.

Actuary

CANDIDATE ID	GENDER	EDUCATION	WORK EXPERIENCE
3cae75a-5f28-44d2-819f-82fa085c5883	f	48 months	201 months
e209730f-e993-4d27-8e9f-730cd3b679d4	m	126 months	181 months
258aeeeb-6738-4232-a390-24ba9a0bc552	m	60 months	230 months
110b0abf-40af-417e-986b-f41c8d274100	f	109 months	128 months
b818ab21-985d-4079-8094-075e8eb3bff4	f	55 months	115 months
ad856ecb-e6db-4411-bc3d-b64c2511067e	m	61 months	85 months
7eec57a5-4400-4ea7-82d6-454ae3296fe7	m	3 months	202 months
460aaf31-9feb-4d5c-afcc-e5d32e31953d	m	3 months	232 months
3a155eb4-fd2a-49b6-88d9-12a9522d4eb9	m	75 months	121 months

☒

CANDIDATE ID	GENDER	EDUCATION	WORK EXPERIENCE
e209730f-e993-4d27-8e9f-730cd3b679d4	m	126 months	181 months
258aeeeb-6738-4232-a390-24ba9a0bc552	m	60 months	230 months
110b0abf-40af-417e-986b-f41c8d274100	f	109 months	128 months
7eec57a5-4400-4ea7-82d6-454ae3296fe7	m	3 months	202 months
b818ab21-985d-4079-8094-075e8eb3bff4	f	55 months	115 months
ad856ecb-e6db-4411-bc3d-b64c2511067e	m	61 months	85 months
3a155eb4-fd2a-49b6-88d9-12a9522d4eb9	m	75 months	121 months
460aaf31-9feb-4d5c-afcc-e5d32e31953d	m	3 months	232 months
3cae75a-5f28-44d2-819f-82fa085c5883	f	48 months	201 months

☐

Next

Additionally, the comparison UI could be configured to collect annotations by asking the user to indicate which returned list of items is more suitable, given the query. In the option for displaying a ranking comparison, the UI will show in the top of the page the query, and below the two rankings to be compared.

The user can interact with the UI in the following ways:

- **Check Box:** depending on the task, the user can choose which ranking is more suitable for the displayed query.
- **Next Button:** the user can navigate to the next query using the Next Button.

For each user, the following interactions can be collected:

- **Best Ranking:** the chosen ranking by the user to be more suitable for the displayed query.

2. Displaying a new Dataset

To use the UI with a new dataset, the following steps should be followed:

1. Under `./datasets`, create a new folder called `<data_name>/data` and save the dataset there. Next, under `./datasets` create a folder called `experiments` in which you can define the query-documents pairs to be displayed when running the UI. More details can be found in the following section: Experiment File.
2. Under `./src/data_readers` create a python file `data_reader_<data_name>.py`. Inside create a class `DataReader<Data_name>`.

```
class DataReader<Data_name>(DataReader):

    def __init__(self, configs):
        super(DataReader<Data_name>,
self).__init__(configs=configs)

    def transform_data(self):
        # read the dataset file from self.data_path
        . . .

        # transform the dataset into two dataframes
        dataset_queries = pd.DataFrame(columns=['title', 'text'])
        dataset_documents = pd.DataFrame(columns=['docID',
'query', 'score', 'group', ...])
        . . .

        # if needed split the dataframe for documents into
        data_train and data_test

        from sklearn.model_selection import train_test_split

        data_train, data_test =
        train_test_split(dataset_documents, test_size=0.2)
        . . .

        # if not needed return None for data_train
        data_test = dataset_documents
        data_train = None

        return dataset_queries, data_train, data_test
```

The `DataReader<Data_name>` should implement the `transform_data` method that should return the following:

- `dataframe_queries` - dataframe describing the queries. It should have the following mandatory columns:
 - `'title'` - this is the title of the query
 - `'text'` - this will be displayed in the UI

For example, if the query is a job description, the `'title'` will be the job title and the `'text'` will be the job description, which includes the job title. If the query is represented by a word/s set both `'title'` and `'text'` to the same value.

- `data_test` - dataframe describing the documents to be displayed.
- `data_train` - dataframe describing the documents to be used when training a ranker or a fairness intervention.

The formatted dataset will be saved under `./datasets/<data_name>/format_data`

3. Under `./configs` create json files following the naming conventions:

- `config_shortlist_<data_name>.json` → to run the Interaction Annotate UI
- `config_compare_<data_name>.json` → to run the Ranking Compare Visualise UI
- `config_compare_annotate_<data_name>.json` → to run the Ranking Compare Annotate UI

in which one should define the configuration to run the tool with. More details about the configuration file can be found in the following section: Configuration File.

3. Configuration File

Under `./configs` create a json file which represents the configurations with which to run the tool with the desired dataset.

`"data_reader_class"` - Configuration for the data.

`"name"` - Specify the dataset name you want to run the tool with.

`"score"` - Column to be used for ranking the documents.

`"group"` - Column to be used by the fairness interventions representing the sensitive information of the documents (e.g. gender).

`"query"` - Column representing the query to which the document is linked to. This should contain the same values found in `dataframe_queries` under `'title'`.

`"docID"` - Column representing the document IDs.

`"ui_display_config"` - Configuration for what to display in the UI.

`"highlight_match"` - Option to highlight the matching words of the query in the item's display field.

`"display_fields"` - List of columns to display.

`"task_description"` - guidelines of how the user should perform the assessment.

`"exit_survey"` - List of questions to be asked in the exit survey.

`"question"` - The question to be displayed.

`"field"` - The field name that will be used to store the value in the database.

`"options"` - If a drop-down is wanted, list of options to be shown in the drop-down, otherwise set the value to `"text"` to display a free text input.

`"mandatory"` - Boolean to be set if the field is mandatory to be completed by the user or not.

Specific fields to declare for the Interaction Annotation UI:

`"shortlist_button"` - Boolean value for whether to display the shortlist button.

`"shortlist_select"` - List representing the range [min, max] of top-k items the user has to select as being the most relevant for the given query.

`"view_button"` - Boolean value for whether to display or not the view button.

`"view_fields"` - List of columns to display when clicking on the view button.

Specific fields to declare for the Score Annotation UI:

`"score_range"` - range for the score bar (e.g [1,5]), with the first value being the start of the range and the second value being the last value from the range. The score bar will have 5 circles starting with values from 1 to 5.

Specific fields to declare for the Ranking Comparison UI:

`"annotate"` - Boolean to be set to true if wanting to run the Ranking Compare Annotate UI, or set to false if wanting to run Ranking Compare Visualise UI.

`"avg_interaction"` - Configuration for the interactions to be displayed.

"experiment_id" - Experiment ID out of which we want to visualise the collected interactions.

"interaction" - Interaction type to be displayed (e.g. "n_views" - clicks on the view button).

"display_metrics" - Configuration for metrics to be displayed.

"top_k" - Compute the metric at @k.

"utility_metrics" - List of utility metrics to be computed and displayed.

"fairness_metrics" - List of fairness metrics to be computed and displayed.

"attention_check" - Configuration to define which task is the attention check.

"task" - The task to be considered as the attention check.

"correct_answer" - The correct answer expected for this task. It should be defined as list of documents for the Interaction Annotation UI. For the Score Annotation UI it should be the value of the correct label.

"iaa" - Configuration for the inter-annotator agreement

"krippendorfs" - Boolean to be set to true to compute Krippendorf's IAA [6]

"cohens" - Boolean to be set to true to compute Cohen's Kappa IAA [5]

"weighted_cohens" - Boolean to be set to true to compute Weighted Cohen's Kappa [7]

"filter_per_task" - Boolean to be set to true if wanting to compute the IAA per task

"train_ranker_config" - Configuration for applying a ranker on the original ranking. If set to null, no ranker will be applied to the original ranking, otherwise, the documents will be ranked based on the predicted score by the defined ranker.

"name" - Name of the ranker to be applied.

"model_path" - Path to load a pre-trained model. If no pre-trained model is available, a new model will be saved at the specified path.

"settings" - List of configurations to run the ranker (specify your own fields used by the ranker class). The ranker class should be initialised with the settings specified.

"ranking_type" - Specify under which tag the ranking of the documents should be saved in the database.

- ranker_<name> - for running a ranking model

"pre/in/post_processing_config" - Configurations to apply a pre-processing fairness intervention on the data. If set to null, no ranker will be applied to the original ranking, otherwise, the documents will be ranked based on the predicted score by the defined ranker.

"name" - Name of the method to be applied.

"model_path" - Path to load a pre-trained model. If no pre-trained model is available, a new model will be saved at the specified path.

"settings" - List of configurations to run the method (specify your own fields used by the fairness intervention class). The fairness intervention class should be initialised with the settings specified.

"ranking_type" - Specify under which tag the ranking of the documents should be saved in the database. It is important to keep the following naming convention for the ranking type:

- preprocessing_<name> - for running a preprocessing fairness intervention.
- postprocessing_<name> - for running a postprocessing fairness intervention.
- inprocessing_<name> - for running an in-processing fairness intervention.

Project Manager
Education Background
Degree: bachelors degree
Major: civil engineering or environmental engineering or management engineering

Professional Experience
Role: project manager
Duration: 2 years

Skills Hard
Agile project management, Lean project management, PRINCE2, MS Project, Health and safety regulation, ISO 9001

Skills Soft
Leadership, Risk management, Coaching, Stakeholder management

NAME	EDUCATION	EXPERIENCE	SKILLS
Manuel Ortega Corral	Degree: Master s Degree	Role: Project manager	Health and safety regulation, ISO 9001, Leadership, Stakeholder management, Lean project management, MS Project, Agile project management, Coaching, Risk management, PRINCE2, Analytics, Incident investigation

Example of the "highlight_match" functionality.

3. Experiment File

Experiment File for the Interaction Annotation UI

"exp_id" - id of the experiment.

"description" - description of the experiment to run.

"tasks" - list of assessments.

 "query_title" - the title of the query to be displayed.

 "setting" - an extra requirement that the user should take into account when doing the assessment.

 "ranking_type" - the order in which to display the ranking of the documents. It should match the "ranking_type" defined in the configuration file.

 "ranking_type" could have the following values:

 "original" - sorted by the "score".

 "ranker_<name>" - sorted by the predicted score of a pre-trained model.

 "pre/in/postprocessing_<name>" - displays the ranking generated by the fairness intervention.

Experiment File for the Score Annotation UI

"exp_id" - id of the experiment.

"description" - description of the experiment to run.

"tasks" - list of assessments.

 "query_title" - the title of the query to be displayed.

 "setting" - an extra requirement that the user should take into account when doing the assessment.

 "index" - the index of the item to be annotated.

Experiment File for the Ranking Comparison UI

"exp_id" - id of the experiment.

"description" - description of the experiment to run.

"tasks" - list of assessments.

 "query_title" - the title of the query to be displayed.

 "ranking_type_1" - the order in which to display the ranking of the documents that is displayed on the left side. It should match the "ranking_type" defined in the configuration file.

 "ranking_type_2" - the order in which to display the ranking of the documents that is displayed on the right side. It should match the "ranking_type" defined in the configuration file.

 "ranking_type" could have the following values:

 "original" - sorted by the "score".

 "ranker_<name>" - sorted by the predicted score of a pre-trained model.

 "pre/in/postprocessing_<name>" - displays the ranking generated by the fairness intervention.

4. Database

The tool works with a MongoDB database. The diagram below describes the collections present in the database. For each dataset, a new database is created automatically together with the required collections. The name of the created dataset will be the name specified in the configuration file as the "data_reader_class". Below the use of each collection is described:

- `documents` - stores the dataframe describing the data to be displayed, meaning the data stored in `data_test`. On top of the fields displayed in the diagram, the tool automatically adds the rest of the columns present in `data_test`. The `preprocessing` field stores a list with preprocessed values given the configurations of the `ranking_type`.

```
_id: ObjectId('65aec8ea0d22641b6cd1e125')
qualification: 0.2688888728417616
gender: "m"
work_experience: 230
edu_experience: 60
hits: 1926
originalQualification: 558540
cid: "24351a23-8157-4bec-b098-710787636d9f"
title: "Actuary"
preprocessing: Array (1)
  0: Object
    ranking_type: "preprocessing_1"
    edu_experience_fair: 19.956521739130395
    work_experience_fair: 230
    hits_fair: 1926
    qualification_fair: 0.2448788728417616
    _cls: "PreDocRepr"
```

- `queries` - stores the dataframe describing the queries, meaning the data stored in `dataframe_queries`.

```
_id: ObjectId('65aec8ea0d22641b6cd1e124')
title: "Actuary"
text: "Actuary"

_id: ObjectId('65b038abf3c7907e51baf8cf')
title: "project_manager"
text: "Project Manager
Education Background
Degree: bachelors degree
Major: civil engineering or environmental engineering or management
Professional Experience
Role: project manager
Duration: 2 years"
```

- `dataset` - stores the query-rankings pairs. For each pair, it stores the id of the query and a list of rankings. Each ranking stores an ordered list of the IDs of the documents linked to this query. The lists are ordered based on the `ranking_type` that was defined in the configuration file.

```
_id: ObjectId('65aec8ec0d22641b6cd1e308')
query: "65aec8ea0d22641b6cd1e109"
rankings: Array (6)
  0: Object
    docs: Array (8)
    ranking_type: "original"
  1: Object
    docs: Array (8)
    ranking_type: "preprocessing_1"
  2: Object
    docs: Array (8)
    ranking_type: "ranker_1:qualification__qualification"
  3: Object
    docs: Array (8)
    ranking_type: "ranker_1:qualification_fair__qualification_fair"
  4: Object
    docs: Array (8)
    ranking_type: "postprocessing_1"
  5: Object
    docs: Array (8)
    ranking_type: "postprocessing_2"
```

- `experiments` - stores the data defined in the experiment file, including the list of tasks to be displayed when running the tool. On the left there is an example belonging to an experiment file used to run the Interaction Annotation UI, while on the right an example belonging to an experiment file used to run the Ranking Compare UI. The corresponding tasks can be seen under the next point: on the left for the Interaction Annotation UI and on the right for Ranking Compare UI.

<pre> _id: ObjectId('65aec8ed0d22641b6cd1e349') _exp_id: "4" _description: "test experiment" tasks: Array (4) 0: "65aec8ed0d22641b6cd1e345" 1: "65aec8ed0d22641b6cd1e346" </pre>	<pre> _id: ObjectId('65aec8ed0d22641b6cd1e344') _exp_id: "3" _description: "compare rankings side by side" tasks: Array (3) 0: "65aec8ed0d22641b6cd1e341" 1: "65aec8ed0d22641b6cd1e342" </pre>
--	--

- `tasks/tasks_compare/tasks_score` - stores the data described in the experiment file for each task, together with the ID of the query-ranking pair stored in the dataset collection. On the left, there is an example belonging to collection `tasks`, while in the middle an example belonging to collection `tasks_compare`, and on the right an example belonging to collection `tasks_score`.

```

_id: ObjectId('65aec8ed0d22641b6cd1e345')
data: "65aec8ec0d22641b6cd1e30b"
ranking_type: "postprocessing_1"
query_title: "Actuary"

```

```

_id: ObjectId('65aec8ed0d22641b6cd1e341')
data: "65aec8ec0d22641b6cd1e30b"
ranking_type_1: "original"
ranking_type_2: "postprocessing_1"
query_title: "Actuary"

```

```

_id: ObjectId('65aec8ed0d22641b6cd1e346')
data: "65aec8ec0d22641b6cd1e30b"
ranking_type: "original"
query_title: "Actuary"

```

```

_id: ObjectId('65aec8ed0d22641b6cd1e342')
data: "65aec8ec0d22641b6cd1e30b"
ranking_type_1: "original"
ranking_type_2: "ranker_1:qualification__qualification"
query_title: "Actuary"

```

```

_id: ObjectId('65c0e3fa193007bae2a8c54b')
data: "65c0df6ad4bfa872c5efdf9"
index: "1"
setting: ""
query_title: "Actuary"
ranking_type: "original"

```

- `user` - stores the ID provided by the user on the first page of the tool and the list of tasks the user interacted with in the tool. As the user interacts with the tool, for each task viewed we store the ID of the tasks together with the list of collected interactions for each document.

```

_id: ObjectId('65aa7c9f40a7cbdf2e56a02e')
_user_id: "1289"
tasks_visited: Array (4)
gender: "Female"
feedback: "Great!"

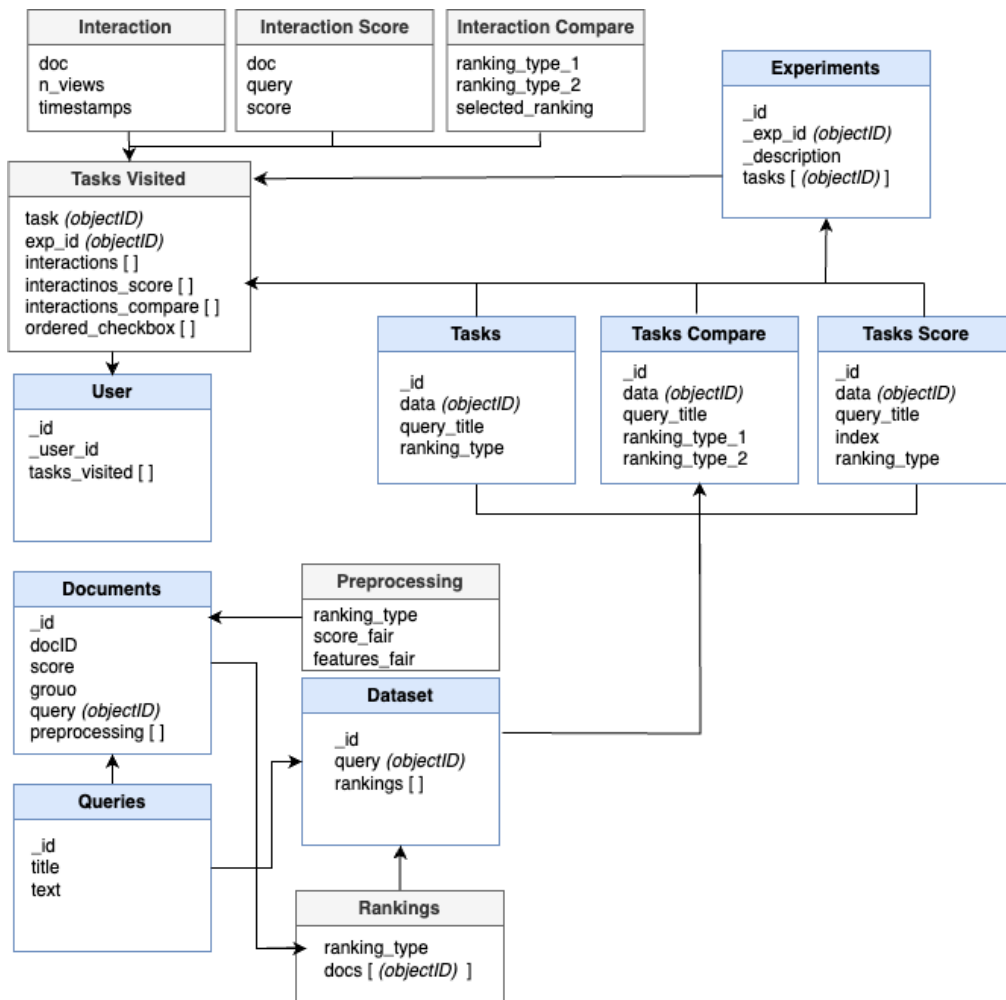
```

Export Data

The collected data can be exported by either running the following command:

`mongoexport --host=<mongo_hostname>:<mongo_port> --collection=<collection_name> --db=<db_name> --out=<output>.json` or from the MongoDB Compass interface¹ as either json or csv files.

¹ <https://www.mongodb.com/products/tools/compass>



Quality of Annotations

Inter-annotator agreement

Anno Rank offers the possibility of computing inter-annotator agreement on the exported annotations. So far, Anno Rank supports computing three IAA's modes (Krippendorff's Alpha [7], Cohen's Kappa [5], Weighted Cohen's Kappa [6]), using the Python *agreement* library². This feature can be activated through a configuration file, for example:

```

"iaa": {
    "krippendorfs": true,
    "cohens": true,
    "weighted_cohens": true,
    "filter_per_task": true }
  
```

, where one has to set the desired IAA metric configuration ("krippendorfs", "cohens", "weighted_cohens") to 'true' for computing the metric or 'false' for not computing the metric.

² <https://pypi.org/project/agreement/>

Enabling "filter_per_task" configuration will group the annotations based on their *experiment_id*, *task_id*, and *doc_id*. If set to false, it will compute the user agreements for every annotating task in the database for a particular dataset.

To compute the IAA metrics for the "shortlisting" feedback collected with the Interaction Annotation UI, the IAA metrics configuration must be included in the *config_shortlist_<dataset_name>.json*. To compute the IAA metrics for the "label" feedback collected with the Score Annotation UI, the IAA metrics configuration should be included in the *config_annotate_score_<dataset_name>.json*.

To generate these metrics, the user can execute this command at any time: `docker exec annorank-container conda run -n tool_ui python /app/src/evaluate/iaa_metrics.py --dataset <dataset_name>`, as long as the MongoDB container is still up. The *<dataset_name>* should be the same as to the `config["data_reader_class"]["name"]` in the same configuration file. After executing the docker exec command above, the output of the IAA calculation will be stored in the */output/iaa_metrics.jsonl*.

Example when "filter_per_task" is True:

```
{
  "annotate": {
    "filters": {
      "task_id": "0",
      "exp_id": "2",
      "doc_id": "6655e6b4e5a0a19c08e948c0"
    },
    "iaa_metrics": {
      "krippendorffs": 0.6,
      "cohens": 0.6,
      "weighted_cohens": 0.6
    },
    "error_message": "No error"
  },
  {
    "ranking": {
      "filters": {
        "task_id": "0",
        "exp_id": "1",
        "doc_id": "6655e6b4e5a0a19c08e948bf"
      },
      "iaa_metrics": {
        "krippendorffs": 0.5,
        "cohens": 0.5,
        "weighted_cohens": 0.5
      },
      "error_message": "No error"
    }
  }
}
```

Example when "filter_per_task" is False:

```
{
  "ranking": {
    "filters": {
      "no_filter": true
    },
    "iaa_metrics": {
      "krippendorffs": 1.0,
      "cohens": 1.0,
      "weighted_cohens": 1.0
    },
    "error_message": "No error"
  }
}
```

The value ranges from 0 to 1, where 0 is perfect disagreement about a specific pair query and document and its annotation score, where 1 is ideal agreement for more than one annotator.

Attention Check

Anno Rank offers the possibility to define an attention check task. In the database the user will have a flag indicating whether the user passed the attention check or not. Using the attention check one could discard the users that did not pass it.

Example of attention check defined for the Flickr dataset:

Interaction Annotate UI:

```
"attention_check": {
  "task": {
    "query_title": "1000344755.jpg",
    "ranking_type": "original"
  },
  "correct_answer": [16, 17, 18]
}
```

Score Annotate UI:

```
"attention_check": {
  "task": {
    "query_title": "1000344755.jpg",
    "index": "2",
    "ranking_type": "original"
  },
  "correct_answer": "Relevant"
}
```

The "correct_answer" indicates the list of documents that the user should shortlist in order to pass the attention check. If the collected feedback from the user differs from the one indicated under "correct_answer" the user will have the attention check failed in the database.

5. Additional Support

5.1 Ranklib Rankers

Ready to Use

The tool offers support for using any model implemented using the ranklib library[3]. It is to be considered that the ranklib library requires java to be installed in the docker container, thus, on the first run it might take longer for the app to start as it first needs to install java. It can be used by defining in the config file the following:

```
"train_ranker_config": {
  "name": "Ranklib",
  "model_path": "./dataset/<data_name>/models/Ranklib", // path to
save/load the model

  "settings": [{
    "features": ["feature_1", "feature_2", "feature_3"], // list of
features to be considered during training
    "pos_th": 0, // threshold to consider relevant documents
    "rel_max": 500, // maximum relevance that is assigned based on
the 'score' column
    "ranker": "RankNet", // ranker name
    "ranker_id": 1, // ranker id

    "metric": "NDCG", // metric to evaluate
    "top_k": 10, // evaluate at top-k
    "lr": 0.000001, // learning rate
    "epochs": 20, //number of epochs

    "train_data":["original"] // define the train data,
    "test_data": ["original"] //define the test data,
    "ranking_type": "ranker_1" ]}]}
```

For more information on how to use ranklib rankers check the ranklib documentation:
<https://sourceforge.net/p/lemur/wiki/RankLib%20How%20to%20use/>.

The "train_data" and "test_data" can be set to the following values:

- "original" - the ground truth is set to be the original score
- <ranking_type> - if the ground truth is set to be the score computed using a fairness intervention

The `train` method saves the input files in the format expected by the ranklib library under `./dataset/<data_name>/models/Ranklib/<train_data>_<test_data>`. The model and predictions are saved under `./dataset/<data_name>/models/Ranklib/<train_data>_<test_data>/ranklib_experiments/<ranker_name>`.

The output of the `predict` method returns a new dataframe that is in the same format as the original one with an appended column representing the predicted score. The column with the predicted relevance should follow the convention `<train_column>__<test_column>`. For example if the train and test data is set to be "original", the predicted score column should be `"score"__"score"`, where `"score"` is the value defined in the config file under the `"score"` field. If the train and test data are pre-processed by a fairness intervention the predicted score column should be `"score"_fair__"score"_fair`.

The new ranking based on the predicted score is saved in the MongoDB database in the collection `dataset`, in the field `rankings`. Given the config presented above, the ranking type of saved in the database will be set to `ranker_1:"score"__"score"`. If a preprocessing fairness intervention is applied on the train/test data, the ranking type saved in the database will be set to `ranker_1:preprocessing_1:"score"_fair__"score"_fair"`

Add a new ranker

In order to add a new ranker the following steps should be followed:

1. Under `./src/rankers` create the following python file: `ranker_<ranker_name>.py`
Inside the python file create the class that implements the ranker method:

```
class <ranker_name>Ranker(Ranker):
    def __init__(self, configs, data_configs,
                 model_path):

        super().__init__(configs, data_configs,
                         model_path)
```

`model_path` - path to save/load the model

`configs` - dictionary of hyperparameters needed to run the ranker. This is defined in the config file as "settings".

`data_configs` - dictionary of configs defined for the `data_reader_class`. This is needed to be able to access the required columns by the fairness intervention.

2. Implement the methods defined in the parent class `Ranker`, which can be found in `./src/fairness_interventions/ranker.py`.

```
def train_model(self, data_train, data_test, experiment)
```

`data_train` - data to train the model
`data_test` - data to evaluate the model during training
`experiment` - tuple containing the `<train_ranking_type>` and `<test_ranking_type>` defined in the configuration file.

The `train` method should save the trained model at the following path:

`self.model_path/<train_ranking_type>__<test_ranking_type>`.

```
def predict(self, data, experiment)
```

`data` - data to run the model on and generate the predicted ranking
`experiment` - tuple containing the `<train_ranking_type>` and `<test_ranking_type>` defined in the configuration file.

The `predict` method should load the model from

`self.model_path/<train_ranking_type>__<test_ranking_type>` and apply it on the `data`.

The method should return a new dataframe that is in the same format as data with an appended column representing the predicted score.

The column with the predicted score should follow the convention

`<train_score_column>__<test_score_column>`.

For example if the `<train_ranking_type>` and

`<test_ranking_type>` is set to be "original", the predicted score column should be "score"__"score". If the train and test data are pre-processed by a fairness intervention the predicted score column should be "score"_fair__"score"_fair, where "score" is the value defined in the config file under the "score" field.

3. If needed any files related to the fairness method can be saved under `./src/rankers/modules/<ranker_name>`
4. Define the new ranker in `./src/constants` in the dictionary containing the rankers.
5. Using the new ranker:

```
"train_ranker_config": {
    "name": "<ranker_name>" // same as the key defined in the
dictionary,
    "model_path":
"./dataset/<data_name>/models/<ranker_name>", // path to
save/load the model
    "settings": [{
        // define any configs needed to run the ranker
        "features": ["feature_1", "feature_2", "feature_3"],
        // list of features to be considered during training
```

```

    "train_data":["original"], // define the train data
    "test_data": ["original"], //define the test data
    "ranking_type": "ranker_1"
  ]}]

```

5.2 Fairness Interventions

Ready to Use

The tool supports applying:

- post-processing fairness intervention: FA*IR[2]. It can be used by defining in the config file the following:

```

"post_processing_config": {
  "name": "FA*IR",
  "model_path": "", //empty as there is no model to save
  "settings": [{
    "k": 10,
    "p": 0.7,
    "alpha": 0.1
    "ranking_type": "postprocessing_1"}]
}

```

- pre-processing fairness intervention: CIF-Rank[1]. It can be used by defining in the config file the following:

```

"pre_processing_config": {
  "name": "CIFRank",
  "model_path": "./dataset/<data_name>/models/CIFRank", // path
  to save/load the model

  "settings": [{
    "pos_th": 0, //threshold to consider positive documents
    "control": "group_value", //control group, the group
    towards which we convert all the data in a counterfactual
    world
    "features": ["field_1", "field_2", "field_3"]//list of
    features (columns from the dataframe) to be considered as
    mediators
  }]
}

```

It is to be considered that the fairness interventions might require additional libraries to be installed in the docker container, thus, on the first run it might take longer for the app to start as it first needs to install the required libraries.

Add a new fairness method

In order to add a new fairness intervention the following steps should be followed:

1. Under ./src/fairnes_interventions create the following python file:
fairness_method_<method_name>.py

Inside the python file create the class that implements the fairness method:

```
class <method_name>(FairnessMethod):  
    def __init__(self, configs, data_configs, model_path):  
        super().__init__(configs, data_configs, model_path)
```

`model_path` - path to save/load the model

`configs` - dictionary of hyperparameters needed to run the fairness method.

This is defined in the config file as "settings".

`data_configs` - dictionary of configs defined for the `data_reader_class`.

This is needed to be able to access the required columns by the fairness intervention.

2. Implement the methods defined in the parent class `FairnessMethod`, which can be found in `./src/fairness_interventions/fairness_method.py`.

```
def train_model(self, data_train)  
    data_train - data to train the model
```

The `train_model` method should save the fairness intervention method to `self.model_path`.

```
def generate_fair_data(self, data):  
    data - data to be used to generate the fair data
```

The `generate_fair_data` method should return a new dataframe that has the same columns as `data`, to which the fair columns are added following the name convention `<column_name>_fair`.

For example, when running a pre-processing fairness intervention like CIF-Rank[1], both the features defined under "features" and the score define under "score" will be changed. The fair columns should be "score_fair", where "score" is the value defined in the config file under the "score" field, and same for all the features defined under "features". When running a post-processing intervention like FA*IR[2], which re-ranks the candidates, then the fair column should represent the new ranking, thus, the name of the returned column is `rank_fair`.

3. If needed any files related to the fairness method can be saved under `./src/fairness_interventions/modules/<method_name>`
4. Define the new fairness method in `./src/constants` in the dictionary containing fairness methods.
5. Using the new fairness method:

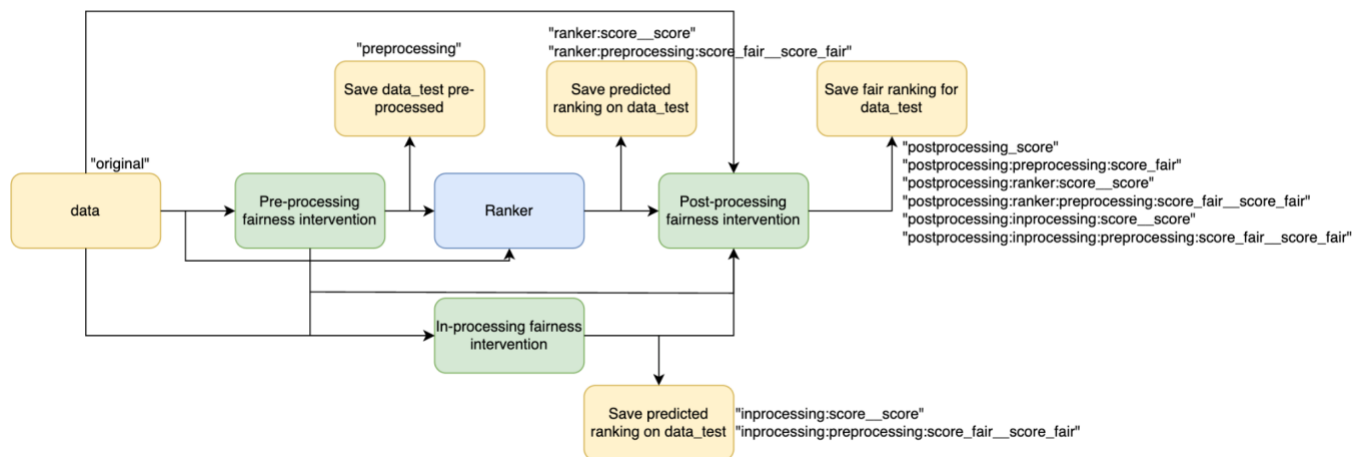
```
"name": "<method_name>" // same as the key defined in the dictionary,  
"model_path": "./dataset/<data_name>/models/<method_name>", // path  
to save/load the model
```

```

"settings": [{
    // define any configs needed to run the fairness method
    "train_data": ["original"], // define the train data for an in-
processing method
    "test_data": ["original"], //define the test data for an in-
processing method
    "ranking_type": // define the ranking type as according to the
naming convention specified in the Configuration File section
}]

```

The figure below describes how the fairness interventions can be applied on the data and how their outputs can be used to train a ranking model, if the configuration file has enabled the use of both the fairness interventions and the use of a ranking model. The **Pre-processing fairness intervention** is applied on the data and saved in the database. As mentioned before, the fair values are saved in the documents collection, while the fair ranking is saved in the dataset collection. The **Ranker**, the **In-processing fairness intervention** and the **Post-processing fairness intervention** can be trained/tested on either the pre-processed data or on the original data. This can be set in the configuration file using the field "train_data" and "test_data". The predicted rankings are saved in the database in the collection `dataset`. The post-processing method can be applied on any kind of ranking, including the ranking based on the original "score" column, the ranking based on the output of the fairness interventions or of the ranker. The predicted rankings are saved in the database under the `dataset` collection.



Given the following example of a configuration for the post-processing intervention using FA*IR[2]:

```

"post_processing_config": {
    "name": "FA*IR",
    "model_path": "",
    "settings": [{
        "k": 10,
        "p": 0.7,
        "alpha": 0.1,

```



```

"test_data": ["original", "preprocessing_1",
"ranker_1:preprocessing_1:qualification_fair__qualification_fair",
"ranker_1:qualification__qualification"],

"ranking_type": "postprocessing_1"}}

```

The post-processing fairness intervention will be applied on the following rankings:

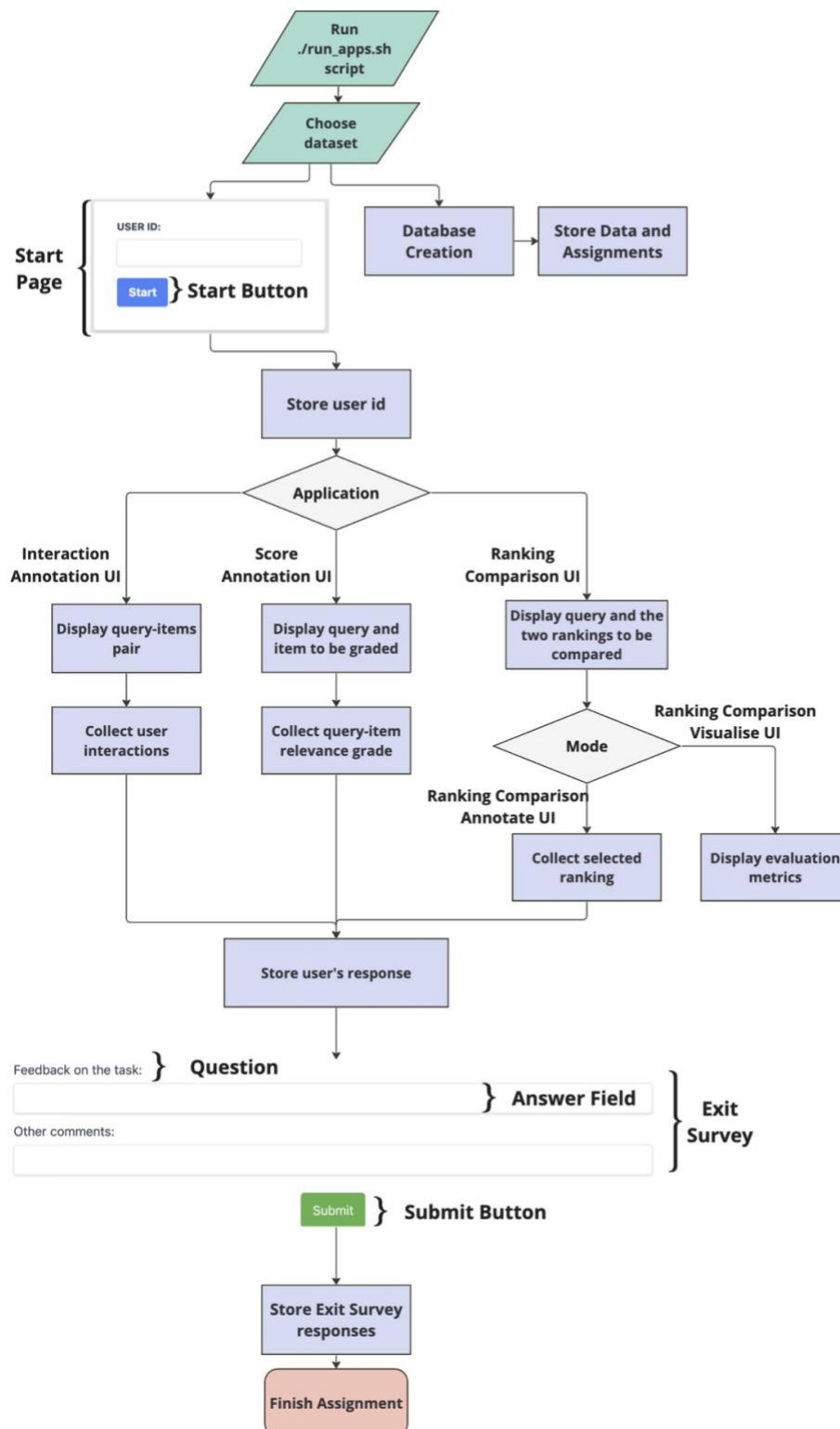
- "original" - the ranking produced by column "score"
- "preprocessing_1" - the ranking produced by the pre-processed score column (qualification_fair)
- "ranker_1:preprocessing_1:qualification_fair__qualification_fair" - the ranking produced by ranker_1 which was trained on the qualification_fair column produced by the pre-processing method defined as preprocessing_1.
- "ranker_1:qualification__qualification" - the ranking produced by ranker_1 which was trained on the qualification column.

The ranking_type of the ranking saved in the database are defined as it follows for the above use cases:

- postprocessing_1:qualification
- postprocessing_1:qualification_fair
- postprocessing_1:ranker_1:preprocessing_1:qualification_fair__qualification_fair
- postprocessing_1:ranker_1:qualification__qualification

6. Run the tool

AnnoRank is designed to be easy to use. We provide easy to run examples, that can be used to tailor the app to specific needs, as well as a step by step tutorial. The figure below depicts the workflow of the AnnoRank application.



Requirements:

AnnoRank utilizes docker containers to handle our applications and database. The *docker-compose* command directs the machine to construct the necessary containers and their dependencies as outlined in the *docker-compose* file. This includes installing the necessary python packages to run the app as well as other requirements such as Java to be able to make use of the Ranklib library, or R to be able to run the pre-processing fairness intervention. Depending on the use of the app one can comment out or add the dependencies in the *docker-compose* file. To make use of other python packages the *env.yml* file needs to be updated. Thus, the AnnoRank web apps can be effortlessly deployed on any web hosting server.

- Install Docker by following the steps presented here: <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04> Select the operating system of the device you want to run the app on and proceed with the steps indicated.
- Install Docker Desktop: <https://www.docker.com/products/docker-desktop/> to be able to monitor the docker instances.
- Install MongoDB Compass: <https://www.mongodb.com/products/tools/compass> to view the dataset created and its collections. The connection should be set as `mongodb://<IP>:27017`. <IP> should be set to IP address for of the machine where the docker container is running.
- If you are using Windows make sure you have WSL2. Allow WSL2 usage in docker settings.

Starting the app is easy and straightforward as one needs to only define the desired configuration files and run the script *run_apps.sh*. Add the paths to the configuration files needed to run the app in: **apps.docker.sh**.

The script will lunch the creation of the docker container, the population of the database with the specified dataset, and the web apps. After starting the app, the web apps will be accessible at the following URLs:

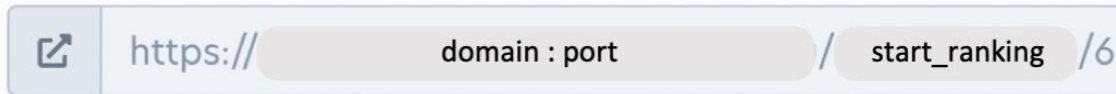
- **Interaction Annotation UI:** http://localhost:5000/start_ranking/<exp_id>
- **Score Annotation UI:** http://localhost:5003/start_annotate/<exp_id>
- **Ranking Comparison Visualise UI:** http://localhost:5001/start_compare/<exp_id>
- **Ranking Comparison Annotate UI:** http://localhost:5002/start_compare_annotate/<exp_id>

The ports where the apps are running are defined in the *docker-compose* file, if needed one can change the ports to other values. The <exp_id> is the ID of the assignment to be displayed to the users which was defined in the experiment file and stored in the collection experiment.

Crowd-sourcing platforms

AnnoRank can be easily used with crowd-sourcing platforms. For example to use AnnoRank with the Prolific platform one needs to only create a study inside the prolific platform and set the URL of the study to the URL of the UI tool:

What is the URL of your study?



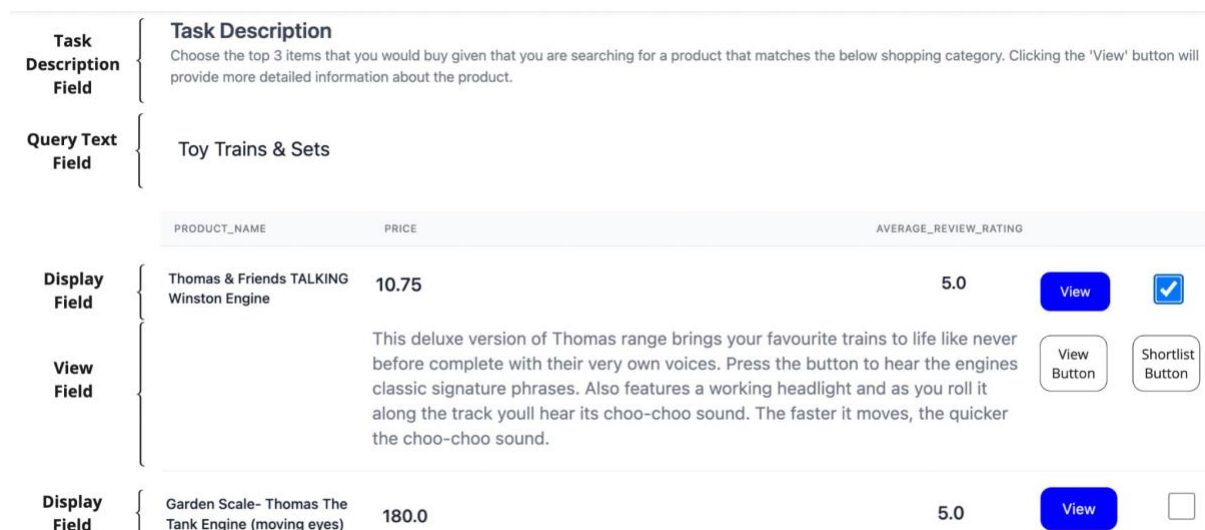
https://domain : port / start_ranking /6

The example above makes use of the Interaction Annotation UI showing to the user the assignment corresponding to ID 6. The user will be registered in the database given the Prolific ID provided by the platform.

7. Ready to run Use Cases

AnnoRank offers tutorials for using pre-built datasets, fairness interventions, and ranker models. However, it can be tailored to meet various information retrieval needs. We describe the three use cases below.

7.1 Retail



PRODUCT_NAME	PRICE	AVERAGE_REVIEW_RATING
Thomas & Friends TALKING Winston Engine	10.75	5.0
Garden Scale- Thomas The Tank Engine (moving eyes)	180.0	5.0

AnnoRank can be valuable in enhancing existing datasets with user interactions that are essential for the training of a recommender system. As a possible concrete use-case for the Interaction Annotation UI, one could consider the collection of user interactions for a retail dataset. Understanding user preferences is essential to improve user satisfaction in the retail

industry. The Amazon e-commerce dataset³ serves as a prominent example of this application. The configuration file depicted in Listing 1 illustrates the straightforward integration of the Amazon dataset features into the AnnoRank UI. For example, the *average_review_rating* is valuable for ranking items in the dataset as it correlates well with user satisfaction.

Additionally, the *amazon_category_and_sub_category* feature is a strong candidate for query due to its association with multiple data points in the dataset. For example, a query for *toys* would retrieve all items related to *toys*. The *manufacturer* feature can be used as a *group* configuration option if fairness to manufacturers is desired, allowing the tool to retrieve items in a rank that is fair to the manufacturing group. For example, not all items from Manufacturer X will always be at the top of the rank; it will also include other manufacturing groups in the top list. By setting *product_name*, *price*, and *average_review_rating* as the *display_field* configuration option, we observe in that these features will be displayed in the *display_field*, while the *product_information* will be presented in the *view_field* that the annotator user will see after they click the *View* button.

Listing 1: Example of configuration file for running the Interaction Annotation UI with the Amazon dataset.

```
{
  "data_reader_class": {
    "name": "amazon",
    "score": "average_review_rating",
    "group": "manufacturer",
    "query": "amazon_category_and_sub_category",
    "docID": "product_name"
  },

  "ui_display_config": {
    "view_button": true,
    "shortlist_button": true,
    "shortlist_select": [3,3],
    "display_fields": ["product_name", "product_description", ...],
    "view_fields": ["product_information"],
    "task_description": "Select top 3 items ...",
  }
}
```

Run the Amazon Use Case

Run the following script and type "amazon":

```
cd UI-Tool-main
./run_apps.sh
```

³ https://www.kaggle.com/code/residentmario/exploring-toy-products-on-amazon/input?select=amazon_e-commerce_sample.csv

Before accessing the links you need to wait for the app to finish the install and start. The Amazon dataset is now available at the following links:

- To access the Interaction Annotation UI go to the following link:
http://localhost:5000/start_ranking/3
- To access the Score Annotate UI go to the following link:
http://localhost:5003/start_annotate/1
- To access the Ranking Comparison Visualise UI go to the following link:
http://localhost:5001/start_compare/2
- To access the Ranking Comparison Annotate UI go to the following link:
http://localhost:5002/start_compare_annotate/2

7.2 Recruitment System

Graded relevance judgments can be applied to both a single dimension or to more dimensions which can be defined according to the features of the dataset. For example, a candidate applying for a job can be evaluated based on three dimensions: the skills, the education, and the work experience required by the job description. Thus, the Score Annotation UI can be customized to a use-case where the user must annotate every defined dimension of the item.

Task Description Field

Task Description
Please pay attention to the extra information provided as it might differ between the tasks. Given the domain displayed below, select a score from 1 to 5 for the given candidate. 1 means that the candidate is not a good fit for the given job, while 5 means that he is a perfect fit. We are looking for a candidate for a senior job in the given domain, that should have a bachelor's degree. EXTRA INFORMATION TO CONSIDER: The employer is an international company.

Query Text Field

Project Manager
Education Background
Degree: bachelors degree
Major: civil engineering or environmental engineering or management engineering

Annotation Field

Manuel Ortega Corral

Work Experience

1 2 3 4 5

ROLE	DURATION (YEARS)	COMPANY
Project manager	1.6 years	Accenture

Education

1 2 3 4 5

DEGREE	UNIVERSITY
Master's Degree	Universidad Carlos III de Madrid
Bachelor's Degree	Universidad Carlos III de Madrid

Skills

1 2 3 4 5

Health and safety regulation	ISO 9001	Leadership	Stakeholder management	Lean project management	MS Project	Agile project management
Coaching	Risk management	PRINCE2	Analytics	Incident investigation		

Overall Score

1 2 3 4 5

Next

The figure above shows an example of how the Score Annotation UI can be customized for a recruitment scenario, where the user is asked to annotate not only the candidate's overall fit to the given job requirements, but also each individual dimension.

The example provided consists of made-up data and does not reflect the data of a real person. In the example provided, when the annotation assessment starts, the user is tasked with evaluating the candidate with respect to the query "Customer Service Representative" and the corresponding job requirements. The user must grade the candidate's skills, education, and work experience given the job requirements. In the example, the "Customer Service Representative" position requires a candidate with at least one year of relevant experience. Additionally, the user must take into account the additional requirement defined in the Task Description field. As mentioned above, such additional requirements can be defined in the experiment configuration file as shown in Listing 2. The user must indicate the relevance of the candidate's experience by selecting a score ranging from 1 (very irrelevant) to 5 (very relevant). Once the user submits their assessment of relevance for all defined dimensions, the feedback is stored in the database and the experiment continues with the next assessment.

Listing 2: Example of configuration experiment file for running the Score Annotation UI with a recruitment dataset.

```
{
  "exp_id": 1,
  "description": "description of the experiment",
  "tasks": [
    {
      "query_title": "customer service representative",
      "setting": "Consider the employer to be an"
      "international company.",
      "ranking_type": "original"
    }, ... ]
}
```

Run the Recruitment Use Case

Run the following script and type "cvs":

```
cd UI-Tool-main
./run_apps.sh
```


Before accessing the links you need to wait for the app to finish the install and start. In order to use the Score Annotate UI with multi dimension annotation adapted for the recruitment use-case, change in "/templates/index_annotate_documents" the line "{% include 'doc_annotate_profile_template.html' %}" with the following: "{% include 'doc_annotate_profile_template_recruitment.html' %}".

The Recruitment dataset is now available at the following links:

- To access the Interaction Annotation UI go to the following link:
http://localhost:5000/start_ranking/1
- To access the Ranking Comparison Visualise UI go to the following link:
http://localhost:5001/start_compare/3

- To access the Ranking Comparison Annotate UI go to the following link:
http://localhost:5002/start_compare_annotate/3
- To access the Score Annotate UI go to the following link:
http://localhost:5003/start_annotate/2

7.3 Multimodal

Task Description Field	<h3>Task Description</h3> <p>Verify whether image and caption given are relevant!</p> <p>Annotate the given caption as either relevant or not relevant for the given image.</p>
Query Text Field	
Display Field	<div>CAPTION</div> <p>A man on a ladder cleans the window of a tall building</p> <div>Choose the label:</div> <div> <input checked="" type="radio"/> Relevant <input type="radio"/> Not Relevant Relevance Label </div> <div> <input type="button" value="Next"/> Navigation Button </div>

Although information retrieval (IR) systems were initially developed for text retrieval, they have since advanced to handle a wide range of multimedia data, which includes text, audio, images, and video. With this in mind, the tool can be used for annotating the relevance of both images and captions. To demonstrate this functionality, we used a sample dataset from Flickr30k⁴, which is used to address the problem of visual notation similarity within linguistic

⁴ <https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset>

expressions captured in the captions provided for individual images \cite{young-etal-2014-image}. The figure above illustrates an example of how the Score Annotation UI can be used to annotate the relevance of a caption to an image query. The image is stored in base64 format in our implementation for easier embedding to the HTML \code{} tag.

Run the Multimodal Use Case

Run the following script and type "flickr":

```
cd UI-Tool-main
./run_apps.sh
```

Before accessing the links you need to wait for the app to finish the install and start.

The Flickr dataset is now available at the following links:

- To access the Interaction Annotation UI go to the following link:
http://localhost:5000/start_ranking/2
- To access the Ranking Comparison Visualise UI go to the following link:
http://localhost:5001/start_compare/3
- To access the Ranking Comparison Annotate UI go to the following link:
http://localhost:5002/start_compare_annotate/3
- To access the Score Annotate UI go to the following link:
http://localhost:5003/start_annotate/1

8. Tutorial

Example on the XING dataset.

1. Download the XING dataset from: https://github.com/MilkaLichtblau/xing_dataset
2. Create the following folder `./datasets/xing/data` and save the dataset there
3. The data reader implemented for this dataset can be found in the following python file `./src/data_readers/data_reader_xing.py`.
4. The experiment files can be found at the following locations:
 - `./datasets/xing/experiments/experiment_shortlist.json` → to run the Interaction Annotation UI
 - `./datasets/xing/experiments/experiment_compare.json` → to run the Ranking Comparison UI
 - `./datasets/xing/experiments/experiment_annotate_score.json` → to run the Score Annotate UI
5. The config files can be found under `./configs/xing_tutorial/`
 - `config_create_db_xing.json` → configuration used to add data in the database the data

The configuration files specifies that a pre-processing fairness intervention and a post-processing fairness intervention is applied on the data. It also specifies to train a ranker on the data and store its prediction in the database. An example of how the data is stored in the database can be seen below.

On the left it can be seen that for each document and for each fairness intervention/ranker the predictions are saved in the database. On the right it can be seen that the ordering of all defined configuration is stored in the `dataset` collection.

- `config_shortlist_xing.json` → to run the Interaction Annotate UI

This configuration file should create the following UI:

In the configuration file, the `"shortlist_button"` is enabled together with the `"view_button"`. The `"shortlist_select"` field defines what is the minimum and the maximum items that the user needs to select as being the top most relevant items given the query. Under the `"display_fields"` you can define which fields to be shown first to the user, while under the `"view_fields"` you can define which fields to be shown when clicking on the "View" button. The `"exit_survey"` field defines what questions the exit survey should contain at the end of the assessments.

- `config_compare_annotate_xing.json` → to run the Ranking Compare Annotate UI

Similarly, one can define which fields to be displayed and what the exit survey should contain. It is important to set "annotate" to be true, otherwise, no user ID will be collected and no checkboxes to select the most suitable ranking given the query and the requirements will not be present.

- `config_compare_xing.json` → to run the Ranking Compare Visualise UI

Similarly, one can define which fields to be displayed. It is important to set "annotate" to false. In the configuration file one can select which metrics to be displayed for each ranking and which interaction data.

- `config_annotate_score_xing.json` → to run the Annotate Score UI

Similarly, one can define which fields to be displayed. It is important to set the range for the scores to be displayed using the field "score_range".

6. Requirements:

Install Docker by following the steps presented here:

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04> Select the operating system of the device you want to run the app on and proceed with the steps indicated.

Install Docker Desktop: <https://www.docker.com/products/docker-desktop/>

Install MongoDB Compass: <https://www.mongodb.com/products/tools/compass> to view the dataset created and its collections. The connection should be set as `mongodb://<IP>:27017`. <IP> should be set to IP address for of the machine where the docker container is running.

7. Run the following script **run_apps.sh**

8. To access the Interaction Annotation UI go to the following link:

http://localhost:5000/start_ranking/4

To access the Ranking Comparison Visualise UI go to the following link:

http://localhost:5001/start_compare/3

To access the Ranking Comparison Annotate UI go to the following link:

http://localhost:5002/start_compare_annotate/3

To access the Score Annotate UI go to the following link:

http://localhost:5003/start_annotate/5

Repository Structure

→ **configs**

- contains examples of configuration files for the supported datasets

→ **dataset**

- contains the supported datasets from the examples provided in the documentation

→ **external_resources**

→ **Anno_Rank_Documentation.pdf**

- the documentation of the tool

→ **Usability_Study_Anno_Rank.pdf**

- the usability study conducted for the tool

→ **flask_app**

→ **app_annotate_document.py**

- runs the Score Annotate UI

→ **app_ranking.py**

- runs the Interaction Annotate UI

→ **app_ranking_compare.py**

- runs the Ranking Comparision UI (researchers only)

→ **app_ranking_compare_annotate.py**

- runs the Ranking Comparison UI for annotators

→ **database.py**

- contains the structure of the database

→ **db_create_pipeline.py**

- contains the script which is adding the formatted data in the database. It runs as well the ranking models and fairness interventions on the data which are saved in the databas

→ **static**

→ **dist**

- contains the supporting css for the UI

→ **js_scripts**

- contains the java scripts code that support the interaction of the user with the UI

→ **templates**

- contains the html files used for rendering the UI
 - the *html files starting with "start"* are used to create the Log-in page for each app
 - *query_template.html* is used for rendering the Query Text Field

- *task_description_template.html* is used for rendering the Task Description Field
- *doc_ranking_display_information_template.html* is used for rendering the list of Display Fields for the UI Interaction App
- *doc_ranking_view_information_template.html* is used for rendering the View Field of the Interaction Annotation UI when a user clicks on the View Button
- *doc_annotate_profile_template.html* is used for rendering the Display Field and Annotation Field of the Score Annotation UI
- *doc_ranking_display_compare_template.html* is used for rendering the Display Field of the Ranking Comparison UI
- the *html files starting with "index"* are used to merge together the Query Field, Task Description Field and the Display Field of each app
- *form_template.html* is used for rendering the exit survey
- *stop_experiment_template.html* is used for rendering the final page of the annotation task

→ src

→ data_readers

- contains the base class used to read the dataset in the format required by AnnoRank

→ evaluate

- contains evaluation scripts for computing utility metrics, fairness metrics and inter-annotator agreement

→ fairness_interventions

- contains the wrapper classes for the supported fairness interventions

→ modules

- contains the supporting scripts for running of the fairness interventions

→ rankers

- contains the wrapper classes for the RankLib library

→modules

- contains the supporting scripts for running of the RankLib library

→ utils

- utility functions

References

- [1] Ke Yang, Joshua R. Loftus, and Julia Stoyanovich. 2021. Causal intersectionality and fair ranking. In Symposium on Foundations of Responsible Computing (FORC).
- [2] Meike Zehlike, Francesco Bonchi, Carlos Castillo, Sara Hajian, Mohamed Megahed, and Ricardo Baeza-Yates. 2017. Fa* ir: A fair top-k ranking algorithm. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. ACM, 1569–1578.
- [3] Dang, V. "The Lemur Project-Wiki-RankLib." Lemur Project,[Online]. Available: <http://sourceforge.net/p/lemur/wiki/RankLib>.
- [4] Van Gysel, Christophe, and Maarten de Rijke. 2018. Pytrec_eval: An extremely fast python interface to trec_eval. The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, 873-876.
- [5] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. Educational and psychological measurement 20.1, 37-46.
- [6] Klaus Krippendorff. 2004. Reliability in content analysis: Some common misconceptions and recommendations. Human communication research 30, 3 (2004), 411–433.
- [7] Joseph Fleiss, Jacob Cohen. 1973. The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. Educational and psychological measurement, 33.3: 613-619.
- [8] Harrisen Scells, Jimmy, & Guido Zuccon. 2021. Big Brother: A Drop-In Website Interaction Logging Service. Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval.