Overview of the TREC 2021 Fair Ranking Track

Michael D. Ekstrand
michaelekstrand@boisestate.edu

Graham McDonald
graham.mcdonald@glasgow.ac.uk

Amifa Raj
amifaraj@u.boisestate.edu

Isaac Johnson
isaac@wikimedia.org

February 22, 2023

1 Introduction

The TREC Fair Ranking Track aims to provide a platform for participants to develop and evaluate novel retrieval algorithms that can provide a fair exposure to a mixture of demographics or attributes, such as ethnicity, that are represented by relevant documents in response to a search query. For example, particular demographics or attributes can be represented by the documents' topical content or authors.

The 2021 Fair Ranking Track adopted a resource allocation task. The task focused on supporting Wikipedia editors who are looking to improve the encyclopedia's coverage of topics under the purview of a WikiProject.[1] WikiProject coordinators and/or Wikipedia editors search for Wikipedia documents that are in need of editing to improve the quality of the article. The 2021 Fair Ranking track aimed to ensure that documents that are about, or somehow represent, certain protected characteristics receive a fair exposure to the Wikipedia editors, so that the documents have an fair opportunity of being improved and, therefore, be well-represented in Wikipedia. The under-representation of particular protected characteristics in Wikipedia can result in systematic biases that can have a negative human, social, and economic impact, particularly for disadvantaged or protected societal groups [3, 5].

2 Task Definition

The 2021 Fair Ranking Track used an *ad hoc* retrieval protocol. Participants were provided with a corpus of documents (a subset of the English language Wikipedia) and a set of queries. A query was of the form of a short list of search terms that represent a WikiProject. Each document in the corpus was relevant to zero to many WikiProjects and associated with zero to many fairness categories.

There were two tasks in the 2021 Fair Ranking Track. In each of the tasks, for a given query, participants were to produce document rankings that are:

1. Relevant to a particular WikiProject.

2. Provide a fair exposure to articles that are associated to particular protected attributes.

The tasks shared a topic set, the corpus, the basic problem structure and the fairness objective. However, they differed in their target user persona, system output (static ranking vs. sequences of rankings) and evaluation metrics. The common problem setup was as follows:

- **Queries** were provided by the organizers and derived from the topics of existing or hypothetical WikiProjects.

---

[1] https://en.wikipedia.org/wiki/WikiProject

1

- **Documents** were Wikipedia articles that may or may not be relevant to any particular WikiProject that is represented by a query.

- **Rankings** were ranked lists of articles for editors to consider working on.

- **Fairness** of exposure was achieved with respect to the **geographic location** of the articles (geographic location annotations were provided). For the evaluation topics, in addition to geographic fairness, to the extent that biographical articles are relevant to the topic, the rankings should have also been fair with respect to an undisclosed **demographic attribute** of the people that the biographies cover, which was gender.

2.1 Task 1: WikiProject Coordinators

The first task focused on WikiProject coordinators as users of the search system; their goal is to search for relevant articles and produce a ranked list of articles needing work that other editors can then consult when looking for work to do.

Output: The output for this task was a **single ranking per query**, consisting of **1000 articles**.

Evaluation was a multi-objective assessment of rankings by the following two criteria:

- Relevance to a WikiProject topic. Relevance assessments were provided for articles for the training queries derived from existing Wikipedia data; evaluation query relevance were assessed by NIST assessors. Ranking relevance was computed with nDCG, using binary relevance and logarithmic decay.

- Fairness with respect to the exposure of different fairness categories in the articles returned in response to a query.

Section 4.2 contains details on the evaluation metrics.

2.2 Task 2: Wikipedia Editors

The second task focused on individual Wikipedia editors looking for work associated with a project. The conceptual model is that rather than maintaining a fixed work list as in Task 1, a WikiProject coordinator would create a saved search, and when an editor looks for work they re-run the search. This means that different editors may receive different rankings for the same query, and differences in these rankings may be leveraged for providing fairness.

Output: The output of this task is **100 rankings per query**, each consisting of **50 articles**.

Evaluation was a multi-objective assessment of rankings by the following three criteria:

- Relevance to a WikiProject topic. Relevance assessments were provided for articles for the training queries derived from existing Wikipedia data; evaluation query relevance was assessed by NIST assessors. Ranking relevance was computed with nDCG.

- Work needed on the article (articles needing more work preferred). We provided the output of an article quality assessment tool for each article in the corpus; for the purposes of this track, we assumed lower-quality articles need more work.

- Fairness with respect to the exposure of different fairness categories in the articles returned in response to a query.

The goal of this task was *not* to be fair to work-needed levels; rather, we consider work-needed and topical relevance to be two components of a multi-objective notion of relevance, so that between two documents with the same topical relevance, the one with more work needed is more relevant to the query in the context of looking for articles to improve.

This task used *expected exposure* to compare the exposure article subjects receive in result rankings to the *ideal* (or *target*) *exposure* they would receive based on their relevance and work-needed [1]. This addresses fundamental limits in the ability to provide fair exposure in a single ranking by examining the exposure over multiple rankings.

For each query, participants provided 100 rankings, which we considered to be samples from the distribution realized by a stochastic ranking policy (given a query q, a distribution π_q over truncated permutations of the documents). Note that this is how we interpret the queries, but it did not mean that a stochastic policy is how the system should have been implemented — other implementation designs were certainly possible. The objective was to provide equitable exposure to documents of comparable relevance and work-needed, aggregated by protected attribute. Section 4.3 has details on the evaluation metrics.

3   Data

This section provides details of the format of the test collection, topics and ground truth. Further details about data generation and limitations can be found in Section 5.2.

3.1   Obtaining the Data

The corpus and query data set is distributed via Globus, and can be obtained in two ways. First, it can be obtained via Globus, from our repository at `https://boi.st/TREC2021Globus`. From this site, you can log in using your institution's Globus account or your own Google account, and synchronize it to your local Globus install or download it with Globus Connect Personal.[2] This method has robust support for restarting downloads and dealing with intermittent connections. Second, it can be downloaded directly via HTTP from: `https://data.boisestate.edu/library/Ekstrand-2021/TRECFairRanking2021/`.

The runs and evaluation qrels will be made available in the ordinary TREC archives.

3.2   Corpus

The corpus consisted of articles from English Wikipedia. We removed all redirect articles, but left the wikitext (markup Wikipedia uses to describe formatting) intact. This was provided as a JSON file, with one record per line, and compressed with gzip (`trec_corpus.json.gz`). Each record contains the following fields:

id The unique numeric Wikipedia article identifier.

title The article title.

url The article URL, to comply with Wikipedia licensing attribution requirements.

text The full article text.

The contents of this corpus were prepared in accordance with, and licensed under, the CC BY-SA 3.0 license.[3] The raw Wikipedia dump files used to produce this corpus are available in the `source` directory; this is primarily for archival purposes, because Wikipedia does not publish dumps indefinitely.

---

[2]`https://www.globus.org/globus-connect-personal`
[3]`https://creativecommons.org/licenses/by-sa/3.0/`

3.3 Topics

Each of the track's training topics is based on a single Wikiproject. The topic is also GZIP-compressed JSON lines (file `trec_topics.json.gz`), with each record containing:

id A query identifier (int)

title The Wikiproject title (string)

keywords A collection of search keywords forming the query text (list of str)

scope A textual description of the project scope, from its project page (string)

homepage The URL for the Wikiproject. This is provided for attribution and not expected to be used by your system as it will not be present in the evaluation data (string)

rel_docs A list of the page IDs of relevant pages (list of int)

The keywords are the primary query text. The scope is there to provide some additional context and potentially support techniques for refining system queries.

In addition to topical relevance, for Task 2: Wikipedia Editors (Section 2.2), participants were also expected to return relevant documents that need more editing work done more highly than relevant documents that need less work done.

3.4 Annotations

NIST assessors annotated the retrieved documents with binary relevance score for given topics. We provided additional options like *unassessable* and *skip* if the document-topic pair is difficult to assess or the assessor is not familiar with the topic. The annotations are incomplete, for reasons including:

- Task 2 requires sequence of rankings which results a large number of dataset, thus it was not possible to annotate all the retrieved documents.

- Some documents were not complete and did not have enough information to match with the topic.

We obtained assessments through tiered pooling, with the goal of having assessments for a coherent subset of rankings that are as complete as possible. We have assessments for the following tiers:

- The first 20 items of all rankings for Task 1 (all queries).

- The first 5 items of the first 25 rankings from every submission to Task 2 (about 75% of the queries).

Details are included with the annotations and metric code.

3.5 Metadata and Fairness Categories

For training data, participants were provided with a geographical fairness ground truth. For the evaluation data, submitted systems were evaluated on how fair their rankings are to the geographical fairness category and an undisclosed personal demographic attribute (gender).

We also provided a simple Wikimedia quality score (a float between 0 and 1 where 0 is no content on the page and 1 is high quality) for optimizing for work-needed in Task 2. Work-needed was operationalized as the reverse—i.e. 1 minus this quality score. The discretized quality scores were used as work-needed for final system evaluation.

This data was provided together in a metadata file (`trec_metadata.json.gz`), in which each line is the metadata for one article represented as a JSON record with the following keys:

page_id Unique page identifier (int)

quality_score Continuous measure of article quality with 0 representing low quality and 1 representing high quality (float in range $[0, 1]$)

quality_score_disc Discrete quality score in which the quality score is mapped to six ordinal categories from low to high: Stub, Start, C, B, GA, FA (string)

geographic_locations Continents that are associated with the article topic. Zero or many of: Africa, Antarctica, Asia, Europe, Latin America and the Caribbean, Northern America, Oceania (list of string)

gender For articles with a gender, the gender of the article's subject, obtained from WikiData.

3.6 Output

For **Task 1**, participants outputted results in rank order in a tab-separated file with two columns:

id The query ID for the topic

page_id ID for the recommended article

For **Task 2**, this file had 3 columns, to account for repeated rankings per query:

id Query ID

rep_number Repeat Number (1-100)

page_id ID for the recommended article

4 Evaluation Metrics

Each task was evaluated with its own metric designed for that task setting. The goal of these metrics was to measure the extent to which a system (1) exposed relevant documents, and (2) exposed those documents in a way that is fair to article topic groups, defined by location (continent) and (when relevant) the gender of the article's subject.

This faces a problem in that Wikipedia itself has well-documented biases: if we target the current group distribution within Wikipedia, we will reward systems that simply reproduce Wikipedia's existing biases instead of promoting social equity. However, if we simply target equal exposure for groups, we would ignore potential real disparities in topical relevance. Due to the biases in Wikipedia's coverage, and the inability to retrieve documents that don't exist to fill in coverage gaps, there is not good empirical data on what the distribution for any particular topic *should* be if systemic biases did not exist in either Wikipedia or society (the "world as it could and should be" [2]). Therefore, in this track we adopted a compromise: we **averaged** the empirical distribution of groups among relevant documents with the world population (for location) or equality (for gender) to derive the target group distribution.

Code to implement the metrics is found at `https://github.com/fair-trec/trec2021-fair-public`.

4.1 Preliminaries

The tasks were to retrieve documents d from a corpus $\mathcal{D}$ that are relevant to a query q. $\mathbf{r}_q \in [0, 1]^{|\mathcal{D}|}$ is a vector of relevance judgements for query q. We denote a ranked list by L; L_i is the document at position i (starting from 1), and L_d^{-1} is the rank of document d. For Task 1, each system returned a single ranked list; for Task 2, it returned a sequence of rankings $\mathcal{L}$.

We represented the group alignment of a document d with an *alignment vector* $\mathbf{a}_d \in [0,1]^{|\mathcal{G}|}$. a_{dg} is document d's alignment with group g. $\mathbf{A} \in [0,1]^{|\mathcal{D}| \times |\mathcal{G}|}$ is the alignment matrix for all documents. $\mathbf{a}_{\mathrm{world}}$ denotes the distribution of the world.[4]

We considered fairness with respect to two group sets, $\mathcal{G}_{\mathrm{geo}}$ and $\mathcal{G}_{\mathrm{gender}}$. We operationalized this intersectional objective by letting $\mathcal{G} = \mathcal{G}_{\mathrm{geo}} \times \mathcal{G}_{\mathrm{gender}}$, the Cartesian product of the two group sets. Further, alignment under either group set may be unknown; we represented this case by treating "unknown" as its own group ($g_?$) in each set. In the product set, a document's alignment may be unknown for either or both groups.

In all metrics, we use **log discounting** to compute attention weights:

$$v_i = \frac{1}{\log_2 \max(i, 2)}$$

Task 2 also considered the work each document needs, represented by $w_d \in \{1, 2, 3, 4\}$.

4.2 Task 1: WikiProject Coordinators (Single Rankings)

For the single-ranking Task 1, we adopted attention-weighted rank fairness (AWRF), first described by Sapiezynski et al. [6] and named by Raj et al. [4]. AWRF computes a vector $\mathbf{d}_L$ of the cumulated exposure a list gives to each group, and a target vector $\mathbf{d}_q^*$; we then compared these with the Jenson-Shannon divergence:

$$\mathbf{d}_L' = \sum_i v_i \mathbf{a}_{L_i} \qquad\qquad \text{cumulated attention}$$

$$\mathbf{d}_L = \frac{\mathbf{d}_L'}{\|\mathbf{d}_L'\|_1} \qquad\qquad \text{normalize to a distribution}$$

$$\mathbf{d}_q^* = \frac{1}{2}\left(\mathbf{A}^{\mathrm{T}}\mathbf{r}_q + \mathbf{a}_{\mathrm{world}}\right)$$

$$\mathrm{AWRF}(L) = 1 - \mathsf{d}_{\mathrm{JS}}(\mathbf{d}_L, \mathbf{d}_q^*) \tag{1}$$

For Task 1, we ignored documents that are fully unknown for the purposes of computing $\mathbf{d}_L$ and $\mathbf{d}_q^*$; they do not contribute exposure to any group.

The resulting metric is in the range $[0, 1]$, with 1 representing a maximally-fair ranking (the distance from the target distribution is minimized). We combined it with an ordinary nDCG metric for utility:

$$\mathrm{NDCG}(L) = \frac{\sum_i v_i r_{qd}}{\mathrm{ideal}} \tag{2}$$

$$M_1(L) = \mathrm{AWRF}(L) \times \mathrm{NDCG}(L) \tag{3}$$

To score well on the final metric M_1, a run must be **both** accurate and fair.

4.3 Task 2: Wikipedia Editors (Multiple Rankings)

For Task 2, we used Expected Exposure [1] to compare the exposure each group receives in the sequence of rankings to the exposure it would receive in a sequence of rankings drawn from an *ideal policy* with the following properties:

- Relevant documents come before irrelevant documents

- Relevant documents are sorted in nonincreasing order of work needed

---

[4]Obtained from `https://en.wikipedia.org/wiki/List_of_continents_and_continental_subregions_by_population`

- Within each work-needed bin of relevant documents, group exposure is fairly distributed according to the average of the distribution of relevant documents and the distribution of global population (the same average target as before).

We have encountered some confusion about whether this task is requiring fairness towards work-needed; as we have designed the metric, work-needed is considered to be a part of (graded) relevance: a document is more relevant if it is relevant to the topic and needs significant work. In the Expected Exposure framework, this combined relevance is used to derive the target policies.

To apply expected exposure, we first define the exposure ϵ_d a document d receives in sequence $\mathcal{L}$:

$$\epsilon_d = \frac{1}{|\mathcal{L}|} \sum_{L \in \mathcal{L}} w_{L_d^{-1}} \tag{4}$$

This forms an exposure vector $\boldsymbol{\epsilon} \in \mathbb{R}^{|\mathcal{D}|}$. It is aggregated into a group exposure vector $\boldsymbol{\gamma}$, including "unknown" as a group:

$$\boldsymbol{\gamma} = \mathbf{A}^\mathrm{T} \boldsymbol{\epsilon} \tag{5}$$

Our implementation rearranges the mean and aggregate operations, but the result is mathematically equivalent.

We then compare these system exposures with the target exposures $\boldsymbol{\epsilon}^*$ for each query. This starts with the per-document ideal exposure; if m_w is the number of relevant documents with work-needed level $w \in \{1, 2, 3, 4\}$, then according to Diaz et al. [1] the ideal exposure for document d is computed as:

$$\epsilon_d^* = \frac{1}{m_{w_d}} \sum_{i=m_{>w_d}+1}^{m_{\geq w_d}} v_i \tag{6}$$

We use this to compute the non-averaged target distribution $\tilde{\boldsymbol{\gamma}}^*$:

$$\tilde{\boldsymbol{\gamma}}^* = \mathbf{A}^\mathrm{T} \boldsymbol{\epsilon}^* \tag{7}$$

Since we include "unknown" as a group, we have a challenge with computing the target distribution by averaging the empirical distribution of relevant documents and the global population — global population does not provide any information on the proportion of relevant articles for which the fairness attributes are relevant. Our solution, therefore, is to average the distribution of *known-group* documents with the world population, and re-normalize so the final distribution is a probability distribution, but derive the proportion of known- to unknown-group documents entirely from the empirical distribution of relevant documents. Extended to handle partially-unknown documents, this procedure proceeds as follows:

- Average the distribution of fully-known documents (both gender and location are known) with the global intersectional population (global population by location and equality by gender).

- Average the distribution of documents with unknown location but known gender with the equality gender distribution.

- Average the distribution of documents with unknown gender but known location with the world population.

The result is the target group exposure $\boldsymbol{\gamma}^*$. We use this to measure the **expected exposure loss**:

$$M_2(\mathcal{L}_q) = \|\boldsymbol{\gamma} - \boldsymbol{\gamma}^*\|_2 \tag{8}$$
$$= \boldsymbol{\gamma} \cdot \boldsymbol{\gamma} - 2\boldsymbol{\gamma} \cdot \boldsymbol{\gamma}^* + \boldsymbol{\gamma}^* \cdot \boldsymbol{\gamma}^*$$
$$\text{EE-D}(\mathcal{L}_q) = \boldsymbol{\gamma}^* \cdot \boldsymbol{\gamma}^* \tag{9}$$
$$\text{EE-R}(\mathcal{L}_q) = \boldsymbol{\gamma} \cdot \boldsymbol{\gamma}^* \tag{10}$$

	nDCG	AWRF	Score	95% CI
UoGTrDExpDisT1	0.2071	0.8299	0.1761	(0.145, 0.212)
UoGTrDRelDiT1	0.2001	0.8072	0.1639	(0.138, 0.193)
UoGTrDivPropT1	0.2157	0.7112	0.1532	(0.128, 0.184)
UoGTrDExpDisLT1	0.1776	0.8197	0.1459	(0.122, 0.173)
RUN1	0.2169	0.6627	0.1425	(0.119, 0.172)
UoGTrRelT1	0.2120	0.6559	0.1373	(0.113, 0.165)
RMITRet	0.2075	0.6413	0.1317	(0.110, 0.159)
1step_pair	0.0838	0.6940	0.0648	(0.046, 0.090)
2step_pair	0.0824	0.6943	0.0638	(0.045, 0.089)
1step_pair_list	0.0820	0.6908	0.0623	(0.045, 0.085)
2step_pair_list	0.0786	0.6912	0.0607	(0.044, 0.083)
RMITRetRerank_1	0.0035	0.6180	0.0026	(0.001, 0.009)
RMITRetRerank_2	0.0035	0.6158	0.0026	(0.001, 0.009)

Table 1: Task 1 runs. Higher score is better (for all metrics).

Lower M_2 is better. It decomposes into two submetrics, the **expected exposure disparity** (EE-D) that measures overall inequality in exposure independent of relevance, for which lower is better; and the **expected exposure relevance** (EE-L) that measures exposure/relevance alignment, for which higher is better [1].

5   Results

This year four different teams submitted a total of 24 runs. All four teams participated in Task 1: Single Rankings (13 runs total), while only three of the four groups participated in Task 2: Multiple Rankings (11 runs total).

5.1   Task 1: WikiProject Coordinators (Single Rankings)

Approaches for Task 1 included:

- RoBERTa model to compute embeddings for text fields.

- A filtering approach to select top ranked documents from either competing rankers or the union of rankers.

- BM25 ranking from pyserini and re-ranked using MMR implicit diversification (without explicit fairness groups). Lambda varied between runs.

- BM25 initial ranking with iterative reranking using fairness calculations to select documents to add to the ranking.

- Relevance ranking using Terrier plus a fairness component that aims to be fair to both the geographic location attribute and an inferred demographic attribute through tailored diversification plus data fusion.

- Optimisation to consider a protected group's distribution in the background collection and the total predicted relevance of the group in the candidate results set.

- Allocating positions in the generated ranking to a protected group proportionally with respect to the total relevance score of the group within the candidate results set.
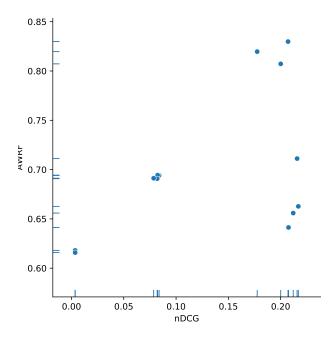
Figure 1: Task 1 submissions by individual component metrics (NDCG and AWRF). Higher values are better for both metrics.

- Relevance-only approaches.

Table 1 shows the submitted systems ranked by the official Task 1 metric M_1 and its component parts nDCG and AWRF. Figure 1 plots the runs with the component metrics on the x and y axes. Notably, each of the approaches from a participating team are clustered in terms of the component metrics and the official M_1 metric.

5.2 Task 2: Wikipedia Editors (Multiple Rankings)

Approaches for Task 2 included:

- A randomized method with BERT and a two-staged Plackett-Luce sampling where relevance scores are combined with work needed.

- An iterative approach that uses RoBERTa and computes a score for each of the top-K documents in the current state, based on the expected exposure of each group so far and the original estimated relevance score, integrating an article's quality score.

- BM25 plus re-ranking iteratively selecting documents by combining relevance, fairness and quality scores.

- Relevance ranking using Terrier plus a fairness component that aims to be fair to both the geographic location attribute and an inferred demographic attribute through tailored diversification plus data fusion to prioritise highly relevant documents while matching the distributions of the protected groups in the generated ranking to their distributions in the background population.

- Minimising the predicted divergence, or skew, in the distributions of the protected groups over all of the rankings within a sequence, compared to the background population.

- Minimising the disparity between a group's expected and actual exposures and learning the importance of the group relevance and background distributions.

	EE-R	EE-D	EE-L	EE-L 95% CI
RUN_task2	9.5508	4.1557	14.9007	(12.303, 19.946)
pl_control_0.6	8.8091	3.2733	15.5017	(12.552, 20.477)
UoGTrRelT2	11.8281	9.4609	15.6514	(13.057, 20.148)
pl_control_0.8	8.6654	3.2550	15.7708	(12.746, 21.251)
pl_control_0.92	8.4802	3.1486	16.0348	(12.820, 21.158)
PL_IRLab_07	5.2790	1.5327	20.8213	(16.283, 28.089)
PL_IRLab_05	4.9331	1.4029	21.3832	(16.579, 28.293)
UoGTrDivPropT2	4.9372	7.1005	27.0726	(21.098, 35.870)
UoGTrDRelDiT2	3.4770	5.5891	28.4816	(22.366, 37.739)
UoGTrDExpDisT2	3.7459	6.1356	28.4903	(22.571, 37.548)
UoGTrLambT2	2.2447	3.4644	28.8216	(22.799, 37.718)

Table 2: Task 2 runs. Lower EE-L is better.

- Relevance-only ranking.

Table 2 shows the submitted systems ranked by the official Task 2 metric EE-L and its component parts EE-D and EE-R. Figure 2 plots the runs with the component metrics on the x and y axes. Overall, the submitted systems generally performed better for one of the component metrics than they did for the other. There are, however, a cluster of four points in Figure 2 that make headway in the trade-off between EE-D and EE-L.

6   Limitations

The data and metrics in this task address a few specific types of unfairness, and do so partially. This is fundamentally true of any fairness intervention, and does not in any way diminish the value of the effort — it is impossible for any data set, task definition, or metric to fully capture fairness in a universal way, and all data and analyses have limitations.

Some of the limitations of the data and task include:

- **Fairness criteria**

  - **Geography**: For each Wikipedia article, we ascertained which, if any, continents are relevant to the content.[5] This was determined by directly looking up several community-maintained (Wikidata) structured data statements about the article. These properties were checked for the presence of countries, which were then mapped to continents via the United Nation's geoscheme.[6] While this data must meet Wikidata's verifiability guidelines,[7] it does suffer from varying levels of incompleteness. For example, only 73% of people on Wikidata have a country of citizenship property.[8] Furthermore, structured data is itself limited—e.g., country of citizenship does not appropriately capture people who are considered stateless though these people may have many strong ties to a country. It is not easy to evaluate whether this data is missing at random or biased against certain regions of the world. Care should be taken when interpreting the absence of associated continents in the data. Further details can be found in the code repository.[9]

---

[5] Code: `https://github.com/geohci/wiki-region-groundtruth/blob/main/wiki-region-data.ipynb`
[6] `https://en.wikipedia.org/wiki/United_Nations_geoscheme`
[7] `https://www.wikidata.org/wiki/Wikidata:Verifiability`
[8] `https://humaniki.wmcloud.org/gender-by-country`
[9] `https://github.com/geohci/wiki-region-groundtruth`
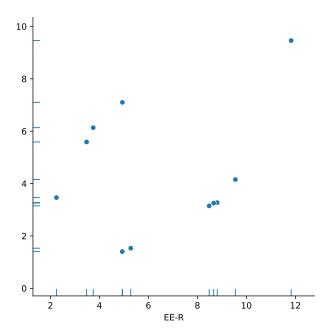
Figure 2: Task 2 submissions by expected exposure subcomponents. Lower EE-D is better; higher EE-R is better.

- **Gender**: For each Wikipedia article, we also ascertained whether it is a biography, and, if so, which gender identity can be associated with the person it is about.[10] This data is also directly determined via Wikidata based on the instance-of property indicating the article is about a human (P31:Q5 in Wikidata terms) and then collecting the value associated with the sex-or-gender property (P21). Coverage here is much higher at 99.98% of biographies on Wikipedia having associated gender data on Wikidata.

  Assigning gender identities to people is not a process without errors, biases, and ethical concerns. Since we are using it to calculate aggregate statistics, we judged it to be less problematic than it would be if we were making decisions about individuals. The process for assigning gender is subject to some community-defined technical limitations[11] and the Wikidata policy on living people[12]. While a separate project, English Wikipedia's policies on gender identity[13] likely inform how many editors handle gender; in particular, this policy explicitly favors the most recent reliably-sourced *self-identification* for gender, so misgendering a biography subject is a violation of Wikipedia policy; there may be erroneous data, but such data seems to be a violation of policy instead of a policy decision. Wikidata:WikiProject LGBT has documented some clear limitations of gender data on Wikidata and a list of further discussions and considerations.[14]

  In our analysis (see Appendix A), we handle nonbinary gender identities by using 4 gender categories: unknown, male, female, and third.

  We advise great care when working with the gender data, particularly outside the immediate context of the TREC task (either its original instance or using the data to evaluate comparable systems).

---

[10] Code: https://github.com/geohci/miscellaneous-wikimedia/blob/master/wikidata-properties-spark/wikidata_gender_information.ipynb

[11] https://www.wikidata.org/wiki/Property_talk:P21#Documentation

[12] https://www.wikidata.org/wiki/Wikidata:Living_people

[13] https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Gender_identity

[14] https://www.wikidata.org/wiki/Wikidata:WikiProject_LGBT/gender

- **Relevance Criteria**

  – **WikiProject Relevance**: For the training queries, relevance was obtained from page lists for existing WikiProjects. While WikiProjects have broad coverage of English Wikipedia and we selected for WikiProjects that had tagged new articles in the recent months in the training data as a proxy for activity, it is certain that almost all WikiProjects are incomplete in tagging relevant content (itself a strong motivation for this task). While it is not easy to measure just how incomplete they are, it should not be assumed that content that has not been tagged as relevant to a WikiProject in the training data is indeed irrelevant.[15] Evaluation query relevance was assessed by NIST assessors, but the large sets of relevant documents and limited budget for working through the pool mean these lists are also incomplete.

  – **Work-needed**: Our proxy for work-needed is a coarse proxy. It is based on just a few simple features (page length, sections, images, and references) and does not reflect the nuances of the work needed to craft a top-quality Wikipedia article.[16] A fully-fledged system for supporting Wikiprojects would also include a more nuanced approach to understanding the work needed for each article and how to appropriately allocate this work.

- **Task Definition**

  – **Existing Article Bias**: The task is limited to topics for which English Wikipedia already has articles. These tasks are not able to counteract biases in the processes by which articles come to exist (or are deleted [7])—recommending articles that should exist but don't is an interesting area for future study.

  – **Fairness constructs**: we focus on gender and geography in this challenge as two metrics for which there is high data coverage and clearer expectations about what "fairer" or more representative coverage might look like. That does not mean these are the most important constructs, but others—e.g., religion, sexuality, culture, race—generally are either more challenging to model or map to fairness goals [5].

References

[1] F. Diaz, B. Mitra, M. D. Ekstrand, A. J. Biega, and B. Carterette. Evaluating stochastic rankings with expected exposure. In *Proc. CIKM '20*, 2020. URL `https://arxiv.org/abs/2004.13157`.

[2] S. Mitchell, E. Potash, S. Barocas, A. D'Amour, and K. Lum. Algorithmic fairness: Choices, assumptions, and definitions. *Annual Review of Statistics and Its Application*, 8, Nov. 2020. doi: 10.1146/annurev-statistics-042720-125902. URL `https://www.annualreviews.org/doi/abs/10.1146/annurev-statistics-042720-125902`.

[3] D. Pedreshi, S. Ruggieri, and F. Turini. Discrimination-aware data mining. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 560–568, 2008.

[4] A. Raj, C. Wood, A. Montoly, and M. D. Ekstrand. Comparing fair ranking metrics. Sept. 2020. URL `http://arxiv.org/abs/2009.01311`.

[5] M. Redi, M. Gerlach, I. Johnson, J. Morgan, and L. Zia. A taxonomy of knowledge gaps for wikimedia projects (first draft). *arXiv preprint arXiv:2008.12314*, 2020.

---

[15]Current Wikiproject tags were extracted from the database tables maintained by the PageAssessments extension: `https://www.mediawiki.org/wiki/Extension:PageAssessments`

[16]For further details, see: `https://meta.wikimedia.org/wiki/Research:Prioritization_of_Wikipedia_Articles/Language-Agnostic_Quality#V1`

[6] P. Sapiezynski, W. Zeng, R. E Robertson, A. Mislove, and C. Wilson. Quantifying the impact of user attentionon fair group representation in ranked lists. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 553–562, 2019.

[7] F. Tripodi. Ms. categorized: Gender, notability, and inequality on wikipedia. *New Media & Society*, page 14614448211023772, 2021.

A Alignments

This appendix provides further details on how the page alignments and target distributions are computed. It is a Jupyter notebook analyzes page alignments and prepares metrics for final use. It needs to be run to create the serialized alignment data files the metrics require; it is available in the code that goes with the appendix.

Its final output is **pickled metric objects**: an instance of the Task 1 and Task 2 metric classes, serialized to a compressed file with binpickle.

A.1 Setup

We begin by loading necessary libraries:

```
from pathlib import Path
import pandas as pd
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gzip
import pickle
import binpickle
from natural.size import binarysize
```

We're going to use ZStandard compression to save our metrics, so let's create a codec object:

```
codec = binpickle.codecs.Blosc('zstd')
```

Set up progress bar and logging support:

```
from tqdm.auto import tqdm
tqdm.pandas(leave=False)

import sys, logging
logging.basicConfig(level=logging.INFO, stream=sys.stderr)
log = logging.getLogger('alignment')
```

Import metric code:

```
%load_ext autoreload
%autoreload 1

%aimport metrics
from trecdata import scan_runs
```

A.2   Loading Data

We first load the page metadata:

```
pages = pd.read_json('data/trec_metadata_eval.json.gz', lines=True)
pages = pages.drop_duplicates('page_id')
pages.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 6023415 entries, 0 to 6023435
Data columns (total 5 columns):
 #   Column                Dtype
---  ------                -----
 0   page_id               int64
 1   quality_score         float64
 2   quality_score_disc    object
 3   geographic_locations  object
 4   gender                object
dtypes: float64(1), int64(1), object(3)
memory usage: 275.7+ MB
```

Now we will load the evaluation topics:

```
eval_topics = pd.read_json('data/eval-topics-with-qrels.json.gz', lines=True)
eval_topics.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49 entries, 0 to 48
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             49 non-null     int64
 1   title          49 non-null     object
 2   rel_docs       49 non-null     object
 3   assessed_docs  49 non-null     object
 4   max_tier       49 non-null     int64
dtypes: int64(2), object(3)
memory usage: 2.0+ KB
```

```
train_topics = pd.read_json('data/trec_topics.json.gz', lines=True)
train_topics.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57 entries, 0 to 56
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        57 non-null     int64
 1   title     57 non-null     object
 2   keywords  57 non-null     object
 3   scope     57 non-null     object
 4   homepage  57 non-null     object
 5   rel_docs  57 non-null     object
dtypes: int64(1), object(5)
memory usage: 2.8+ KB
```

Train and eval topics use a disjoint set of IDs:

```
train_topics['id'].describe()
```

```
count    57.000000
mean     29.000000
std      16.598193
min       1.000000
25%      15.000000
50%      29.000000
75%      43.000000
max      57.000000
Name: id, dtype: float64
```

```
eval_topics['id'].describe()
```

```
count     49.000000
mean     125.346939
std       14.687794
min      101.000000
25%      113.000000
50%      125.000000
75%      138.000000
max      150.000000
Name: id, dtype: float64
```

This allows us to create a single, integrated topics list for convenience:

```
topics = pd.concat([train_topics, eval_topics], ignore_index=True)
topics['eval'] = False
topics.loc[topics['id'] >= 100, 'eval'] = True
topics.head()
```

```
   id         title                                   keywords  \
0   1   Agriculture  [agriculture, crops, livestock, forests, farming]
1   2  Architecture  [architecture, skyscraper, landscape, building...
2   3     Athletics       [athletics, player, sports, game, gymnastics]
3   4      Aviation  [aviation, aircraft, airplane, airship, pilot,...
4   5      Baseball                                        [baseball]

                                       scope  \
0  This WikiProject strives to develop and improv...
1  This WikiProject aims to: 1. Thoroughly explor...
2  WikiProject Athletics, a project focused on im...
3  The project generally considers any article re...
4  Articles pertaining to baseball including base...

                                    homepage  \
0  https://en.wikipedia.org/wiki/Wikipedia:WikiPr...
1  https://en.wikipedia.org/wiki/Wikipedia:WikiPr...
2  https://en.wikipedia.org/wiki/Wikipedia:WikiPr...
3  https://en.wikipedia.org/wiki/Wikipedia:WikiPr...
4  https://en.wikipedia.org/wiki/Wikipedia:WikiPr...
```

```
                                      rel_docs assessed_docs  max_tier  \
0  [572, 627, 903, 1193, 1542, 1634, 3751, 3866, ...       NaN       NaN
1  [682, 954, 1170, 1315, 1322, 1324, 1325, 1435,...       NaN       NaN
2  [5729, 8490, 9623, 10391, 12231, 13791, 16078,...       NaN       NaN
3  [849, 852, 1293, 1902, 1942, 2039, 2075, 2082,...       NaN       NaN
4  [1135, 1136, 1293, 1893, 2129, 2140, 3797, 380...       NaN       NaN

    eval
0  False
1  False
2  False
3  False
4  False
```

Finally, a bit of hard-coded data - the world population:

```python
world_pop = pd.Series({
    'Africa': 0.155070563,
    'Antarctica': 1.54424E-07,
    'Asia': 0.600202585,
    'Europe': 0.103663858,
    'Latin America and the Caribbean': 0.08609797,
    'Northern America': 0.049616733,
    'Oceania': 0.005348137,
})
world_pop.name = 'geography'
```

And a gender global target:

```python
gender_tgt = pd.Series({
    'female': 0.495,
    'male': 0.495,
    'third': 0.01
})
gender_tgt.name = 'gender'
gender_tgt.sum()
```

```
1.0
```

Xarray intesectional global target:

```python
geo_tgt_xa = xr.DataArray(world_pop, dims=['geography'])
gender_tgt_xa = xr.DataArray(gender_tgt, dims=['gender'])
int_tgt = geo_tgt_xa * gender_tgt_xa
int_tgt
```

```
<xarray.DataArray (geography: 7, gender: 3)>
array([[7.67599287e-02, 7.67599287e-02, 1.55070563e-03],
       [7.64398800e-08, 7.64398800e-08, 1.54424000e-09],
       [2.97100280e-01, 2.97100280e-01, 6.00202585e-03],
       [5.13136097e-02, 5.13136097e-02, 1.03663858e-03],
       [4.26184951e-02, 4.26184951e-02, 8.60979700e-04],
```

```
        [2.45602828e-02, 2.45602828e-02, 4.96167330e-04],
        [2.64732781e-03, 2.64732781e-03, 5.34813700e-05]])
Coordinates:
  * geography  (geography) object 'Africa' 'Antarctica' ... 'Oceania'
  * gender     (gender) object 'female' 'male' 'third'
```

And the order of work-needed codes:

```
work_order = [
    'Stub',
    'Start',
    'C',
    'B',
    'GA',
    'FA',
]
```

Now all our background data is set up.

A.3 Query Relevance

We now need to get the qrels for the topics. This is done by creating frames with entries for every relevant document; missing documents are assumed irrelevant (0).

In the individual metric evaluation files, we will truncate each run to only the assessed documents (with a small amount of noise), so this is a safe way to compute.

First the training topics:

```
train_qrels = train_topics[['id', 'rel_docs']].explode('rel_docs', ignore_index=True)
train_qrels.rename(columns={'rel_docs': 'page_id'}, inplace=True)
train_qrels['page_id'] = train_qrels['page_id'].astype('i4')
train_qrels = train_qrels.drop_duplicates()
train_qrels.head()

    id  page_id
0   1      572
1   1      627
2   1      903
3   1     1193
4   1     1542

eval_qrels = eval_topics[['id', 'rel_docs']].explode('rel_docs', ignore_index=True)
eval_qrels.rename(columns={'rel_docs': 'page_id'}, inplace=True)
eval_qrels['page_id'] = eval_qrels['page_id'].astype('i4')
eval_qrels = eval_qrels.drop_duplicates()
eval_qrels.head()

     id  page_id
0   101      915
1   101     2948
2   101     9110
3   101     9742
4   101    10996
```

And concatenate:

```
qrels = pd.concat([train_qrels, eval_qrels], ignore_index=True)
```

A.4   Page Alignments

All of our metrics require page "alignments": the protected-group membership of each page.

A.4.1   Geography

Let's start with the straight page geography alignment for the public evaluation of the training queries. The page metadata has that; let's get the geography column.

```
page_geo = pages[['page_id', 'geographic_locations']].explode('geographic_locations', ignore_index=True)
page_geo.head()
```

```
   page_id geographic_locations
0       12                  NaN
1       25                  NaN
2       39                  NaN
3      290                  NaN
4      303      Northern America
```

And we will now pivot this into a matrix so we get page alignment vectors:

```
page_geo_align = page_geo.assign(x=1).pivot(index='page_id', columns='geographic_locations', values='x')
page_geo_align.rename(columns={np.nan: 'Unknown'}, inplace=True)
page_geo_align.fillna(0, inplace=True)
page_geo_align.head()
```

```
geographic_locations  Unknown  Africa  Antarctica  Asia  Europe  \
page_id
12                        1.0     0.0         0.0   0.0     0.0
25                        1.0     0.0         0.0   0.0     0.0
39                        1.0     0.0         0.0   0.0     0.0
290                       1.0     0.0         0.0   0.0     0.0
303                       0.0     0.0         0.0   0.0     0.0


geographic_locations  Latin America and the Caribbean  Northern America  \
page_id
12                                                0.0               0.0
25                                                0.0               0.0
39                                                0.0               0.0
290                                               0.0               0.0
303                                               0.0               1.0


geographic_locations  Oceania
page_id
12                        0.0
25                        0.0
39                        0.0
290                       0.0
303                       0.0
```

And convert this to an xarray for multidimensional usage:

```
page_geo_xr = xr.DataArray(page_geo_align, dims=['page', 'geography'])
page_geo_xr
```

```
<xarray.DataArray (page: 6023415, geography: 8)>
array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.]])
Coordinates:
  * page       (page) int64 12 25 39 290 ... 67268663 67268668 67268699 67268751
  * geography  (geography) object 'Unknown' 'Africa' ... 'Oceania'

binarysize(page_geo_xr.nbytes)

'385.50 MiB'
```

A.4.2 Gender

The "undisclosed personal attribute" is gender. Not all articles have gender as a relevant variable - articles not about a living being generally will not.

We're going to follow the same approach for gender:

```
page_gender = pages[['page_id', 'gender']].explode('gender', ignore_index=True)
page_gender.fillna('unknown', inplace=True)
page_gender.head()
```

```
   page_id   gender
0       12  unknown
1       25  unknown
2       39  unknown
3      290  unknown
4      303  unknown
```

We need to do a little targeted repair - there is an erroneous record of a gender of "Taira no Kiyomori" is actually male. Replace that:

```
page_gender = page_gender.loc[page_gender['gender'] != 'Taira no Kiyomori']
```

Now, we're going to do a little more work to reduce the dimensionality of the space. Points:

1. Trans men are men
2. Trans women are women
3. Cisgender is an adjective that can be dropped for the present purposes

The result is that we will collapse "transgender female" and "cisgender female" into "female".

The **downside** to this is that trans men are probably significantly under-represented, but are now being collapsed into the dominant group.

```
pgcol = page_gender['gender']
pgcol = pgcol.str.replace(r'(?:tran|ci)sgender\s+((?:fe)?male)', r'\1', regex=True)
```

Now, we're going to group the remaining gender identities together under the label 'third'. As noted above, this is a debatable exercise that collapses a lot of identity.

```python
genders = ['unknown', 'male', 'female', 'third']
pgcol[~pgcol.isin(genders)] = 'third'
```

Now put this column back in the frame and deduplicate.

```python
page_gender['gender'] = pgcol
page_gender = page_gender.drop_duplicates()
```

And make an alignment matrix (reordering so 'unknown' is first for consistency):

```python
page_gend_align = page_gender.assign(x=1).pivot(index='page_id', columns='gender', values='x')
page_gend_align.fillna(0, inplace=True)
page_gend_align = page_gend_align.reindex(columns=['unknown', 'female', 'male', 'third'])
page_gend_align.head()
```

```
gender   unknown  female  male  third
page_id
12           1.0     0.0   0.0    0.0
25           1.0     0.0   0.0    0.0
39           1.0     0.0   0.0    0.0
290          1.0     0.0   0.0    0.0
303          1.0     0.0   0.0    0.0
```

Let's see how frequent each of the genders is:

```python
page_gend_align.sum(axis=0).sort_values(ascending=False)
```

```
gender
unknown    4246540.0
male       1441813.0
female      334946.0
third          452.0
dtype: float64
```

And convert to an xarray:

```python
page_gend_xr = xr.DataArray(page_gend_align, dims=['page', 'gender'])
page_gend_xr
```

```
<xarray.DataArray (page: 6023415, gender: 4)>
array([[1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       ...,
       [0., 1., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.]])
Coordinates:
  * page     (page) int64 12 25 39 290 ... 67268663 67268668 67268699 67268751
  * gender   (gender) object 'unknown' 'female' 'male' 'third'
```

```python
binarysize(page_gend_xr.nbytes)
```

```
'192.75 MiB'
```

A.4.3 Intersectional Alignment

We'll now convert this data array to an **intersectional** alignment array:

```
page_xalign = page_geo_xr * page_gend_xr
page_xalign

<xarray.DataArray (page: 6023415, geography: 8, gender: 4)>
array([[[1., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        ...,
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[1., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        ...,
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[1., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        ...,
...
        ...,
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[1., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        ...,
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[1., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        ...,
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]])
Coordinates:
  * page       (page) int64 12 25 39 290 ... 67268663 67268668 67268699 67268751
  * geography  (geography) object 'Unknown' 'Africa' ... 'Oceania'
  * gender     (gender) object 'unknown' 'female' 'male' 'third'
```

```
binarysize(page_xalign.nbytes)
```

'1.54 GiB'

Make sure that did the right thing and we have intersectional numbers:

```
page_xalign.sum(axis=0)
```

```
<xarray.DataArray (geography: 8, gender: 4)>
array([[2.06922e+06, 8.21940e+04, 4.05772e+05, 1.85000e+02],
       [7.76580e+04, 1.04830e+04, 4.34670e+04, 8.00000e+00],
       [9.62500e+03, 0.00000e+00, 1.00000e+00, 0.00000e+00],
       [4.27422e+05, 3.79980e+04, 1.35310e+05, 2.10000e+01],
       [7.65203e+05, 9.67970e+04, 4.27747e+05, 6.30000e+01],
       [1.01464e+05, 1.61660e+04, 6.77640e+04, 4.00000e+00],
       [7.21244e+05, 8.25430e+04, 3.30205e+05, 1.59000e+02],
       [9.26820e+04, 1.45240e+04, 5.07260e+04, 2.00000e+01]])
Coordinates:
  * geography  (geography) object 'Unknown' 'Africa' ... 'Oceania'
  * gender     (gender) object 'unknown' 'female' 'male' 'third'
```

And make sure combination with targets work as expected:

```
(page_xalign.sum(axis=0) + int_tgt) * 0.5
```

```
<xarray.DataArray (geography: 7, gender: 3)>
array([[5.24153838e+03, 2.17335384e+04, 4.00077535e+00],
       [3.82199400e-08, 5.00000038e-01, 7.72120000e-10],
       [1.89991486e+04, 6.76551486e+04, 1.05030010e+01],
       [4.83985257e+04, 2.13873526e+05, 3.15005183e+01],
       [8.08302131e+03, 3.38820213e+04, 2.00043049e+00],
       [4.12715123e+04, 1.65102512e+05, 7.95002481e+01],
       [7.26200132e+03, 2.53630013e+04, 1.00000267e+01]])
Coordinates:
  * geography  (geography) object 'Africa' 'Antarctica' ... 'Oceania'
  * gender     (gender) object 'female' 'male' 'third'
```

A.5 Task 1 Metric Preparation

Now that we have our alignments and qrels, we are ready to prepare the Task 1 metrics.

Task 1 ignores the "unknown" alignment category, so we're going to create a `kga` frame (for **K**nown **G**eographic **A**lignment), and corresponding frames for intersectional alignment.

```
page_kga = page_geo_align.iloc[:, 1:]
page_kga.head()
```

```
geographic_locations  Africa  Antarctica  Asia  Europe  \
page_id
12                       0.0         0.0   0.0     0.0
25                       0.0         0.0   0.0     0.0
39                       0.0         0.0   0.0     0.0
290                      0.0         0.0   0.0     0.0
303                      0.0         0.0   0.0     0.0
```

```
geographic_locations  Latin America and the Caribbean  Northern America  \
page_id
12                                                 0.0               0.0
25                                                 0.0               0.0
39                                                 0.0               0.0
290                                                0.0               0.0
303                                                0.0               1.0


geographic_locations  Oceania
page_id
12                        0.0
25                        0.0
39                        0.0
290                       0.0
303                       0.0
```

Intersectional is a little harder to do, because things can be **intersectionally unknown**: we may know gender but not geography, or vice versa. To deal with these missing values for Task 1, we're going to ignore *totally unknown* values, but keep partially-known as a category.

We also need to ravel our tensors into a matrix for compatibility with the metric code. Since 'unknown' is the first value on each axis, we can ravel, and then drop the first column.

```
xshp = page_xalign.shape
xshp = (xshp[0], xshp[1] * xshp[2])
page_xa_df = pd.DataFrame(page_xalign.values.reshape(xshp), index=page_xalign.indexes['page'])
page_xa_df.head()
```

```
        0    1    2    3    4    5    6    7    8    9  ...   22   23   24  \
page                                                   ...
12    1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
25    1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
39    1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
290   1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
303   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  1.0


       25   26   27   28   29   30   31
page
12    0.0  0.0  0.0  0.0  0.0  0.0  0.0
25    0.0  0.0  0.0  0.0  0.0  0.0  0.0
39    0.0  0.0  0.0  0.0  0.0  0.0  0.0
290   0.0  0.0  0.0  0.0  0.0  0.0  0.0
303   0.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 32 columns]
```

And drop unknown, to get our page alignment vectors:

```
page_kia = page_xa_df.iloc[:, 1:]
```

A.5.1 Geographic Alignment

We'll start with the metric configuration for public training data, considering only geographic alignment. We configure the metric to do this for both the training and the eval queries.

Training Queries

```
train_qalign = train_qrels.join(page_kga, on='page_id').drop(columns=['page_id']).groupby('id').sum()
tqa_sums = train_qalign.sum(axis=1)
train_qalign = train_qalign.divide(tqa_sums, axis=0)

train_qalign.head()
```

	Africa	Antarctica	Asia	Europe	Latin America and the Caribbean	\
id						
1	0.049495	0.00000	0.121886	0.356566	0.031650	
2	0.013388	0.00000	0.112008	0.574026	0.026105	
3	0.109664	0.00000	0.125529	0.456033	0.100040	
4	0.062495	0.00025	0.116161	0.327272	0.079514	
5	0.000835	0.00000	0.065433	0.010149	0.064755	

	Northern America	Oceania
id		
1	0.261616	0.178788
2	0.228715	0.045758
3	0.158419	0.050316
4	0.369277	0.045032
5	0.850192	0.008636

```
train_qtarget = (train_qalign + world_pop) * 0.5
train_qtarget.head()
```

	Africa	Antarctica	Asia	Europe	\
id					
1	0.102283	7.721200e-08	0.361044	0.230115	
2	0.084229	7.721200e-08	0.356105	0.338845	
3	0.132367	7.721200e-08	0.362866	0.279848	
4	0.108783	1.250113e-04	0.358182	0.215468	
5	0.077953	7.721200e-08	0.332818	0.056906	

	Latin America and the Caribbean	Northern America	Oceania
id			
1	0.058874	0.155616	0.092068
2	0.056101	0.139166	0.025553
3	0.093069	0.104018	0.027832
4	0.082806	0.209447	0.025190
5	0.075427	0.449904	0.006992

And we can prepare a metric and save it:

```
t1_train_metric = metrics.Task1Metric(train_qrels.set_index('id'), page_kga, train_qtarget)
binpickle.dump(t1_train_metric, 'task1-train-geo-metric.bpk', codec=codec)

INFO:binpickle.write:pickled 337312647 bytes with 5 buffers
```

Eval Queries Do the same thing for the eval data for a geo-only eval metric:

```
eval_qalign = eval_qrels.join(page_kga, on='page_id').drop(columns=['page_id']).groupby('id').sum()
eqa_sums = eval_qalign.sum(axis=1)
eval_qalign = eval_qalign.divide(eqa_sums, axis=0)
eval_qtarget = (eval_qalign + world_pop) * 0.5
t1_eval_metric = metrics.Task1Metric(eval_qrels.set_index('id'), page_kga, eval_qtarget)
binpickle.dump(t1_eval_metric, 'task1-eval-geo-metric.bpk', codec=codec)

INFO:binpickle.write:pickled 337312643 bytes with 5 buffers
```

A.5.2 Intersectional Alignment

Now we need to apply similar logic, but for the intersectional (geography * gender) alignment.

As noted as above, we need to carefully handle the unknown cases.

Demo To demonstrate how the logic works, let's first work it out in cells for one query (1).

What are its documents?

```
qdf = qrels[qrels['id'] == 1]
qdf.name = 1
qdf
```

```
      id    page_id
0      1        572
1      1        627
2      1        903
3      1       1193
4      1       1542
...   ..        ...
6959   1   67066971
6960   1   67075177
6961   1   67178925
6962   1   67190032
6963   1   67244439

[6964 rows x 2 columns]
```

We can use these page IDs to get its alignments:

```
q_xa = page_xalign.loc[qdf['page_id'].values, :, :]
q_xa
```

```
<xarray.DataArray (page: 6964, geography: 8, gender: 4)>
array([[[1., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        ...,
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[1., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
```

```
          ...,
         [0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[1., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.],
          ...,
...
          ...,
         [0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[1., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.],
          ...,
         [0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[1., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.],
          ...,
         [0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]]])
Coordinates:
  * page       (page) int64 572 627 903 1193 ... 67178925 67190032 67244439
  * geography  (geography) object 'Unknown' 'Africa' ... 'Oceania'
  * gender     (gender) object 'unknown' 'female' 'male' 'third'
```

Summing over the first axis ('page') will produce an alignment matrix:

```
q_am = q_xa.sum(axis=0)
q_am

<xarray.DataArray (geography: 8, gender: 4)>
array([[3767.,   52.,  200.,    0.],
       [ 128.,   12.,    7.,    0.],
       [   0.,    0.,    0.,    0.],
       [ 322.,   11.,   29.,    0.],
       [ 940.,   23.,   96.,    0.],
       [  79.,    8.,    7.,    0.],
       [ 618.,   28.,  131.,    0.],
       [ 484.,    6.,   41.,    0.]])
Coordinates:
  * geography  (geography) object 'Unknown' 'Africa' ... 'Oceania'
  * gender     (gender) object 'unknown' 'female' 'male' 'third'
```

Now we need to do reset the (0,0) coordinate (full unknown), and normalize to a proportion.

```
q_am[0, 0] = 0
q_am = q_am / q_am.sum()
q_am

<xarray.DataArray (geography: 8, gender: 4)>
array([[0.        , 0.01613904, 0.06207325, 0.        ],
       [0.03972688, 0.00372439, 0.00217256, 0.        ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.09993793, 0.00341403, 0.00900062, 0.        ],
       [0.29174426, 0.00713842, 0.02979516, 0.        ],
       [0.02451893, 0.00248293, 0.00217256, 0.        ],
       [0.19180633, 0.00869025, 0.04065798, 0.        ],
       [0.15021726, 0.0018622 , 0.01272502, 0.        ]])
Coordinates:
  * geography  (geography) object 'Unknown' 'Africa' ... 'Oceania'
  * gender     (gender) object 'unknown' 'female' 'male' 'third'
```

Ok, now we have to - very carefully - average with our target modifier. There are three groups:

- known (use intersectional target)
- known-geo (use geo target)
- known-gender (use gender target)

For each of these, we need to respect the fraction of the total it represents. Let's compute those fractions:

```
q_fk_all = q_am[1:, 1:].sum()
q_fk_geo = q_am[1:, :1].sum()
q_fk_gen = q_am[:1, 1:].sum()
q_fk_all, q_fk_geo, q_fk_gen

(<xarray.DataArray ()>
 array(0.12383613),
 <xarray.DataArray ()>
 array(0.79795158),
 <xarray.DataArray ()>
 array(0.07821229))
```

And now do some surgery. Weighted-average to incorporate the target for fully-known:

```
q_tm = q_am.copy()
q_tm[1:, 1:] *= 0.5
q_tm[1:, 1:] += int_tgt * 0.5 * q_fk_all
q_tm

<xarray.DataArray (geography: 8, gender: 4)>
array([[0.00000000e+00, 1.61390441e-02, 6.20732464e-02, 0.00000000e+00],
       [3.97268777e-02, 6.61502352e-03, 5.83910794e-03, 9.60166894e-05],
       [0.00000000e+00, 4.73300933e-09, 4.73300933e-09, 9.56163501e-11],
       [9.99379268e-02, 2.01028882e-02, 2.28961843e-02, 3.71633817e-04],
       [2.91744258e-01, 6.74645100e-03, 1.80748185e-02, 6.41866532e-05],
       [2.45189323e-02, 3.88031961e-03, 3.72513649e-03, 5.33101956e-05],
       [1.91806331e-01, 5.86585240e-03, 2.18497134e-02, 3.07217202e-05],
```

```
        [1.50217256e-01, 1.09501611e-03, 6.52642517e-03, 3.31146285e-06]])
Coordinates:
  * geography  (geography) object 'Unknown' 'Africa' ... 'Oceania'
  * gender     (gender) object 'unknown' 'female' 'male' 'third'
```

And for known-geo:

```
q_tm[1:, :1] *= 0.5
q_tm[1:, :1] += geo_tgt_xa * 0.5 * q_fk_geo
```

And known-gender:

```
q_tm[:1, 1:] *= 0.5
q_tm[:1, 1:] += gender_tgt_xa * 0.5 * q_fk_gen
```

```
q_tm
```

```
<xarray.DataArray (geography: 8, gender: 4)>
array([[0.00000000e+00, 2.74270639e-02, 5.03941651e-02, 3.91061453e-04],
        [8.17328395e-02, 6.61502352e-03, 5.83910794e-03, 9.60166894e-05],
        [6.16114376e-08, 4.73300933e-09, 4.73300933e-09, 9.56163501e-11],
        [2.89435265e-01, 2.01028882e-02, 2.28961843e-02, 3.71633817e-04],
        [1.87231499e-01, 6.74645100e-03, 1.80748185e-02, 6.41866532e-05],
        [4.66104719e-02, 3.88031961e-03, 3.72513649e-03, 5.33101956e-05],
        [1.15699041e-01, 5.86585240e-03, 2.18497134e-02, 3.07217202e-05],
        [7.72424054e-02, 1.09501611e-03, 6.52642517e-03, 3.31146285e-06]])
Coordinates:
  * geography  (geography) object 'Unknown' 'Africa' ... 'Oceania'
  * gender     (gender) object 'unknown' 'female' 'male' 'third'
```

Now we can unravel this and drop the first entry:

```
q_tm.values.ravel()[1:]
```

```
array([2.74270639e-02, 5.03941651e-02, 3.91061453e-04, 8.17328395e-02,
        6.61502352e-03, 5.83910794e-03, 9.60166894e-05, 6.16114376e-08,
        4.73300933e-09, 4.73300933e-09, 9.56163501e-11, 2.89435265e-01,
        2.01028882e-02, 2.28961843e-02, 3.71633817e-04, 1.87231499e-01,
        6.74645100e-03, 1.80748185e-02, 6.41866532e-05, 4.66104719e-02,
        3.88031961e-03, 3.72513649e-03, 5.33101956e-05, 1.15699041e-01,
        5.86585240e-03, 2.18497134e-02, 3.07217202e-05, 7.72424054e-02,
        1.09501611e-03, 6.52642517e-03, 3.31146285e-06])
```

Implementation   Now, to do this for every query, we'll use a function that takes a data frame for a query's relevant docs and performs all of the above operations:

```
def query_xalign(qdf):
    pages = qdf['page_id']
    pages = pages[pages.isin(page_xalign.indexes['page'])]
    q_xa = page_xalign.loc[pages.values, :, :]
    q_am = q_xa.sum(axis=0)

    # clear and normalize
    q_am[0, 0] = 0
```

```
    q_am = q_am / q_am.sum()

    # compute fractions in each section
    q_fk_all = q_am[1:, 1:].sum()
    q_fk_geo = q_am[1:, :1].sum()
    q_fk_gen = q_am[:1, 1:].sum()

    # known average
    q_am[1:, 1:] *= 0.5
    q_am[1:, 1:] += int_tgt * 0.5 * q_fk_all

    # known-geo average
    q_am[1:, :1] *= 0.5
    q_am[1:, :1] += geo_tgt_xa * 0.5 * q_fk_geo

    # known-gender average
    q_am[:1, 1:] *= 0.5
    q_am[:1, 1:] += gender_tgt_xa * 0.5 * q_fk_gen

    # and return the result
    return pd.Series(q_am.values.ravel()[1:])

query_xalign(qdf)

0      2.742706e-02
1      5.039417e-02
2      3.910615e-04
3      8.173284e-02
4      6.615024e-03
5      5.839108e-03
6      9.601669e-05
7      6.161144e-08
8      4.733009e-09
9      4.733009e-09
10     9.561635e-11
11     2.894353e-01
12     2.010289e-02
13     2.289618e-02
14     3.716338e-04
15     1.872315e-01
16     6.746451e-03
17     1.807482e-02
18     6.418665e-05
19     4.661047e-02
20     3.880320e-03
21     3.725136e-03
22     5.331020e-05
23     1.156990e-01
24     5.865852e-03
25     2.184971e-02
26     3.072172e-05
27     7.724241e-02
```

```
28    1.095016e-03
29    6.526425e-03
30    3.311463e-06
dtype: float64
```

Now with that function, we can compute the alignment vector for each query.

```
train_qtarget = train_qrels.groupby('id').apply(query_xalign)
train_qtarget
```

```
           0         1         2         3         4         5         6  \
id
1   0.027427  0.050394  0.000391  0.081733  0.006615  0.005839  0.000096
2   0.012235  0.032073  0.000232  0.073168  0.003571  0.003669  0.000070
3   0.022553  0.035541  0.000292  0.023527  0.040981  0.059556  0.000574
4   0.012472  0.029112  0.000209  0.094840  0.004409  0.004901  0.000086
5   0.023416  0.063398  0.000436  0.020521  0.024932  0.025194  0.000504
6   0.126820  0.201558  0.001691  0.000021  0.028097  0.030215  0.000519
7   0.050837  0.115432  0.000836  0.000026  0.034747  0.051416  0.000646
8   0.038785  0.054361  0.000521  0.000065  0.038044  0.038746  0.000702
9   0.059002  0.157276  0.001087  0.005630  0.028051  0.056632  0.000554
10  0.064617  0.137545  0.001016  0.046254  0.008038  0.008038  0.000162
11  0.060435  0.128320  0.000979  0.029760  0.020073  0.021687  0.000358
12  0.020151  0.038214  0.000332  0.007651  0.032456  0.032725  0.000653
13  0.014332  0.035379  0.000250  0.009309  0.034996  0.038045  0.000655
14  0.046850  0.041965  0.000561  0.041140  0.016396  0.016052  0.000299
15  0.068267  0.151259  0.001103  0.000377  0.030622  0.032842  0.000601
16  0.135784  0.156886  0.001904  0.009884  0.025627  0.025196  0.000448
17  0.067934  0.174287  0.001217  0.013029  0.026528  0.051011  0.000503
18  0.018251  0.036588  0.000276  0.030822  0.023188  0.027944  0.000445
19  0.103573  0.097443  0.001247  0.013579  0.027784  0.025663  0.000485
20  0.089270  0.071413  0.000807  0.038786  0.021002  0.020420  0.000334
21  0.108101  0.211342  0.001605  0.027217  0.016228  0.017338  0.000296
22  0.026161  0.059192  0.000429  0.003891  0.036267  0.043268  0.000671
23  0.024937  0.066835  0.000461  0.027385  0.025748  0.048173  0.000503
24  0.081419  0.177767  0.001498  0.068779  0.005003  0.005585  0.000101
25  0.107975  0.121840  0.001155  0.008671  0.026618  0.026711  0.000511
26  0.041788  0.043538  0.000835  0.068601  0.027419  0.033208  0.000323
27  0.038810  0.078021  0.000587  0.055195  0.006552  0.006769  0.000132
28  0.023529  0.051249  0.000376  0.000296  0.035349  0.035468  0.000714
29  0.087081  0.191848  0.001402  0.031724  0.014463  0.016746  0.000289
30  0.056876  0.090125  0.000739  0.018007  0.027345  0.030689  0.000509
31  0.053926  0.104654  0.000819  0.058785  0.010576  0.010732  0.000158
32  0.061350  0.076521  0.003383  0.000257  0.036019  0.035636  0.001173
33  0.030389  0.053928  0.000424  0.061496  0.010157  0.009832  0.000194
34  0.096244  0.136311  0.001169  0.022726  0.020403  0.022574  0.000389
35  0.087547  0.111057  0.000998  0.017422  0.028479  0.026493  0.000460
36  0.112926  0.177999  0.001500  0.033846  0.017653  0.017997  0.000286
37  0.186952  0.406140  0.003590  0.019620  0.006586  0.008204  0.000132
38  0.027033  0.049546  0.000385  0.018166  0.026696  0.030492  0.000534
39  0.050328  0.062821  0.000569  0.013145  0.027890  0.028400  0.000558
40  0.165541  0.274278  0.002251  0.035258  0.004510  0.005636  0.000086
```

```
41  0.043914  0.084869  0.000647  0.009875  0.035748  0.047432  0.000617
42  0.042045  0.077639  0.000601  0.072132  0.011772  0.012999  0.000206
43  0.088816  0.236394  0.001634  0.045119  0.006863  0.007995  0.000139
44  0.073139  0.110720  0.000924  0.008159  0.029961  0.033199  0.000677
45  0.102279  0.175641  0.001615  0.010408  0.025755  0.026896  0.000464
46  0.049735  0.096882  0.000772  0.057043  0.016799  0.021288  0.000285
47  0.002770  0.006868  0.000048  0.065464  0.006435  0.006435  0.000126
48  0.048879  0.130381  0.000901  0.000374  0.031816  0.055275  0.000632
49  0.023329  0.043920  0.000338  0.008751  0.032725  0.037509  0.000638
50  0.021747  0.035992  0.000290  0.006747  0.053491  0.085334  0.000706
51  0.042260  0.067426  0.000551  0.013784  0.027519  0.028445  0.000553
52  0.011937  0.020075  0.000161  0.067402  0.003922  0.003828  0.000077
53  0.064740  0.088902  0.000772  0.019943  0.029669  0.032655  0.000507
54  0.006491  0.008134  0.000073  0.071422  0.004520  0.004449  0.000084
55  0.001431  0.003218  0.000023  0.106429  0.000485  0.000485  0.000010
56  0.127312  0.042703  0.000891  0.003167  0.049226  0.030645  0.000626
57  0.064556  0.101994  0.000905  0.243893  0.072830  0.147165  0.000380

                 7              8              9   ...         21        22  \
id                                                 ...
1    6.161144e-08   4.733009e-09   4.733009e-09   ...   0.003725   0.000053
2    6.684003e-08   3.431828e-09   3.431828e-09   ...   0.003276   0.000039
3    1.665751e-08   2.774295e-08   2.774295e-08   ...   0.039630   0.000312
4    1.197781e-04   4.231748e-09   4.231748e-09   ...   0.002998   0.000048
5    2.031701e-08   2.482831e-08   2.482831e-08   ...   0.038382   0.000280
6    2.098131e-11   2.559433e-08   5.057750e-06   ...   0.019846   0.000288
7    2.510823e-11   3.182076e-08   3.182076e-08   ...   0.052265   0.000358
8    6.442770e-11   3.460808e-08   3.460808e-08   ...   0.029176   0.000396
9    5.304181e-09   2.728670e-08   2.728670e-08   ...   0.074846   0.000307
10   4.535438e-08   8.004062e-09   8.004062e-09   ...   0.005030   0.000090
11   2.692777e-08   1.763909e-08   1.763909e-08   ...   0.015362   0.000199
12   7.618901e-09   3.220520e-08   3.220520e-08   ...   0.018456   0.000363
13   8.094806e-09   3.230352e-08   3.230352e-08   ...   0.042302   0.000364
14   4.055074e-08   1.473139e-08   1.473139e-08   ...   0.009435   0.000166
15   2.900724e-10   2.964395e-08   2.964395e-08   ...   0.031470   0.000334
16   9.842701e-09   2.208922e-08   2.208922e-08   ...   0.015761   0.000249
17   8.276972e-09   2.481865e-08   2.481865e-08   ...   0.024976   0.000280
18   2.861625e-08   2.194840e-08   2.194840e-08   ...   0.034100   0.000247
19   1.328731e-08   2.391224e-08   2.391224e-08   ...   0.018990   0.000269
20   3.147228e-08   1.646900e-08   1.646900e-08   ...   0.009959   0.000185
21   2.290303e-08   1.461251e-08   1.461251e-08   ...   0.012144   0.000165
22   3.776095e-09   3.307219e-08   3.307219e-08   ...   0.056006   0.000373
23   1.946374e-06   2.469539e-08   2.469539e-08   ...   0.062443   0.000278
24   9.717212e-05   4.981659e-09   4.981659e-09   ...   0.004040   0.000056
25   8.449778e-09   2.520965e-08   2.520965e-08   ...   0.019174   0.000284
26   3.841889e-08   1.590954e-08   1.590954e-08   ...   0.018563   0.000179
27   5.496504e-08   6.524548e-09   6.524548e-09   ...   0.040466   0.000073
28   2.952129e-10   3.520143e-08   3.520143e-08   ...   0.020403   0.000396
29   2.675860e-08   1.426019e-08   1.426019e-08   ...   0.010663   0.000161
30   1.508983e-08   2.510388e-08   2.510388e-08   ...   0.023381   0.000283
31   4.918211e-08   7.782549e-09   7.782549e-09   ...   0.006588   0.000088
```

```
32   2.559296e-10   3.269452e-08   3.269452e-08   ...   0.028172   0.000623
33   5.137857e-08   9.548780e-09   9.548780e-09   ...   0.006866   0.000108
34   2.046956e-08   1.915463e-08   1.915463e-08   ...   0.014686   0.000216
35   1.596396e-08   2.268898e-08   2.268898e-08   ...   0.022650   0.000256
36   2.620009e-08   1.407441e-08   1.407441e-08   ...   0.011137   0.000159
37   1.795991e-08   6.524601e-09   6.524601e-09   ...   0.003671   0.000073
38   1.809025e-08   2.632373e-08   2.632373e-08   ...   0.060619   0.000296
39   1.283586e-08   2.751992e-08   2.751992e-08   ...   0.017383   0.000310
40   3.449979e-08   4.246660e-09   4.246660e-09   ...   0.004435   0.000048
41   3.226594e-06   3.024027e-08   3.024027e-08   ...   0.044898   0.000341
42   4.738518e-08   1.016696e-08   1.016696e-08   ...   0.011024   0.000115
43   4.527317e-04   6.834258e-09   6.834258e-09   ...   0.005395   0.000077
44   7.089076e-09   2.764847e-08   2.764847e-08   ...   0.021775   0.000311
45   9.390308e-09   2.288793e-08   2.288793e-08   ...   0.021510   0.000258
46   1.188359e-05   1.402732e-08   1.402732e-08   ...   0.020366   0.000158
47   6.389264e-08   6.222844e-09   6.222844e-09   ...   0.004587   0.000070
48   3.726048e-10   3.114976e-08   3.114976e-08   ...   0.040424   0.000351
49   8.473119e-09   3.144257e-08   3.144257e-08   ...   0.043209   0.000354
50   4.998116e-09   3.352802e-08   3.352802e-08   ...   0.052002   0.000378
51   1.364949e-08   2.725022e-08   2.725022e-08   ...   0.026850   0.000307
52   6.702715e-08   3.811868e-09   3.811868e-09   ...   0.002125   0.000043
53   1.477752e-08   2.500337e-08   2.500337e-08   ...   0.033812   0.000282
54   6.771162e-08   4.140914e-09   4.140914e-09   ...   0.003063   0.000055
55   1.841679e-02   4.832648e-10   4.832648e-10   ...   0.000407   0.000005
56   2.393092e-09   3.050337e-08   3.050337e-08   ...   0.017079   0.000351
57   3.318478e-08   1.539335e-08   1.539335e-08   ...   0.008785   0.000173
```

	23	24	25	26	27	28	29 \
id							
1	0.115699	0.005866	0.021850	0.000031	0.077242	0.001095	0.006526
2	0.114341	0.003410	0.015195	0.000022	0.021868	0.000564	0.001981
3	0.024684	0.023416	0.049637	0.000202	0.006253	0.007383	0.012547
4	0.176399	0.003847	0.020421	0.000027	0.020782	0.000413	0.002940
5	0.121270	0.014276	0.274932	0.000161	0.002668	0.000967	0.002729
6	0.000052	0.043116	0.107266	0.000206	0.000011	0.006256	0.011167
7	0.000078	0.019805	0.095237	0.000212	0.000025	0.005422	0.022694
8	0.000206	0.078311	0.129290	0.000390	0.000009	0.006967	0.010040
9	0.018965	0.013625	0.092816	0.000177	0.003089	0.002289	0.014736
10	0.035855	0.009666	0.019314	0.000052	0.002990	0.000561	0.000703
11	0.092015	0.019263	0.108094	0.000114	0.007596	0.002075	0.005840
12	0.051786	0.059223	0.374352	0.000248	0.000264	0.001192	0.001423
13	0.013124	0.018640	0.052865	0.000210	0.001067	0.003577	0.008987
14	0.181400	0.066251	0.040491	0.000096	0.013693	0.007532	0.004746
15	0.000563	0.027242	0.081673	0.000192	0.000053	0.002905	0.005808
16	0.036754	0.055762	0.113471	0.000143	0.001202	0.001196	0.004210
17	0.004093	0.008305	0.012349	0.000161	0.009967	0.006974	0.068220
18	0.031099	0.013915	0.022385	0.000142	0.006749	0.004953	0.012171
19	0.039630	0.060017	0.046816	0.000155	0.005411	0.009550	0.007900
20	0.095128	0.046829	0.029166	0.000301	0.013124	0.022892	0.022115
21	0.057092	0.021791	0.063754	0.000095	0.005012	0.001616	0.003392
22	0.003974	0.021571	0.041587	0.000215	0.002399	0.004103	0.008146

```
23   0.015388   0.009643   0.019771   0.000162   0.003743   0.001768   0.005251
24   0.027928   0.001698   0.004029   0.000032   0.003765   0.000173   0.000173
25   0.014254   0.033132   0.029596   0.000164   0.003736   0.006643   0.002734
26   0.082215   0.031768   0.030960   0.000373   0.011697   0.001897   0.001897
27   0.078969   0.010545   0.020511   0.000042   0.002120   0.000443   0.000226
28   0.000095   0.013312   0.015522   0.000228   0.000010   0.001309   0.001398
29   0.042854   0.018284   0.065101   0.000093   0.002069   0.000779   0.001065
30   0.054574   0.052398   0.122136   0.000180   0.004532   0.005387   0.012916
31   0.219361   0.016084   0.041382   0.000051   0.012882   0.002229   0.002363
32   0.000847   0.114151   0.153417   0.007096   0.000136   0.007762   0.010439
33   0.201573   0.036510   0.035536   0.000062   0.012494   0.003009   0.002928
34   0.052654   0.036538   0.053567   0.000124   0.002378   0.004169   0.005004
35   0.041752   0.047621   0.046893   0.000147   0.011016   0.005620   0.002706
36   0.078388   0.029082   0.049013   0.000091   0.007908   0.004810   0.004007
37   0.006714   0.002602   0.003479   0.000042   0.000689   0.000226   0.000293
38   0.093509   0.024950   0.162515   0.000171   0.003899   0.002351   0.008503
39   0.055119   0.113381   0.161825   0.000179   0.006309   0.002483   0.004523
40   0.012825   0.001876   0.003391   0.000028   0.001379   0.000311   0.000700
41   0.005666   0.024614   0.043943   0.000203   0.003246   0.008020   0.014145
42   0.110739   0.015094   0.044887   0.000066   0.023176   0.002026   0.007717
43   0.114799   0.004912   0.021888   0.000044   0.028936   0.000689   0.007027
44   0.019854   0.057218   0.108907   0.000179   0.002327   0.006161   0.009630
45   0.028069   0.064359   0.057022   0.000257   0.005433   0.007857   0.009324
46   0.076146   0.016152   0.056417   0.000127   0.007826   0.001007   0.002357
47   0.323062   0.012990   0.052856   0.000040   0.006125   0.000402   0.000588
48   0.001728   0.012689   0.107462   0.000202   0.000013   0.002687   0.050811
49   0.012655   0.025667   0.059400   0.000204   0.004836   0.009386   0.031794
50   0.012840   0.029417   0.052854   0.000244   0.001221   0.007420   0.012814
51   0.043987   0.026819   0.034925   0.000177   0.003715   0.004418   0.006425
52   0.025983   0.001793   0.003307   0.000025   0.003268   0.000132   0.000700
53   0.015227   0.032331   0.037326   0.000162   0.009199   0.014358   0.015661
54   0.230042   0.010873   0.017122   0.000043   0.016006   0.000788   0.000843
55   0.169513   0.000705   0.002079   0.000003   0.054992   0.000017   0.000154
56   0.008593   0.124741   0.009902   0.000342   0.000587   0.015458   0.001078
57   0.012217   0.006500   0.007919   0.000100   0.001217   0.000533   0.000533

          30
id
1    3.311463e-06
2    2.401088e-06
3    1.941043e-05
4    2.960754e-06
5    1.737119e-05
6    2.797145e-05
7    2.226348e-05
8    3.745849e-05
9    1.909121e-05
10   5.600064e-06
11   1.234124e-05
12   2.253246e-05
13   2.260124e-05
```

```
14   1.030686e-05
15   2.074047e-05
16   1.545478e-05
17   1.736444e-05
18   1.535626e-05
19   1.673026e-05
20   1.152258e-05
21   1.022368e-05
22   2.313905e-05
23   1.727819e-05
24   3.485431e-06
25   1.763800e-05
26   1.113115e-05
27   4.564918e-06
28   2.462878e-05
29   9.977184e-06
30   1.756400e-05
31   5.445081e-06
32   9.152766e-04
33   6.680830e-06
34   1.340159e-05
35   1.587440e-05
36   9.847201e-06
37   4.564955e-06
38   1.841747e-05
39   2.742263e-04
40   2.971187e-06
41   2.115769e-05
42   7.113344e-06
43   4.781607e-06
44   1.934433e-05
45   7.035552e-05
46   9.814252e-06
47   4.353830e-06
48   2.179402e-05
49   2.199888e-05
50   2.345797e-05
51   1.906569e-05
52   2.666984e-06
53   1.749368e-05
54   1.075735e-05
55   3.381175e-07
56   4.295529e-05
57   1.077000e-05

[57 rows x 31 columns]
```

And save:

```
t1_train_metric = metrics.Task1Metric(train_qrels.set_index('id'), page_kia, train_qtarget)
binpickle.dump(t1_train_metric, 'task1-train-metric.bpk', codec=codec)

INFO:binpickle.write:pickled 1493808204 bytes with 5 buffers
```

Do the same for eval:

```
eval_qtarget = eval_qrels.groupby('id').apply(query_xalign)
t1_eval_metric = metrics.Task1Metric(eval_qrels.set_index('id'), page_kia, eval_qtarget)
binpickle.dump(t1_eval_metric, 'task1-eval-metric.bpk', codec=codec)

INFO:binpickle.write:pickled 1493808200 bytes with 5 buffers
```

A.6   Task 2 Metric Preparation

Task 2 requires some different preparation.

We're going to start by computing work-needed information:

```
page_work = pages.set_index('page_id').quality_score_disc.astype(pd.CategoricalDtype(ordered=True))
page_work = page_work.cat.reorder_categories(work_order)
page_work.name = 'quality'
```

A.6.1   Work and Target Exposure

The first thing we need to do to prepare the metric is to compute the work-needed for each topic's pages, and use that to compute the target exposure for each (relevant) page in the topic.

This is because an ideal ranking orders relevant documents in decreasing order of work needed, followed by irrelevant documents. All relevant documents at a given work level should receive the same expected exposure.

First, look up the work for each query page ('query page work', or qpw):

```
qpw = qrels.join(page_work, on='page_id')
qpw
```

```
            id    page_id  quality
0            1        572        C
1            1        627       FA
2            1        903        C
3            1       1193        B
4            1       1542       GA
...        ...        ...      ...
2199072    150   63656179    Start
2199073    150   63807245      NaN
2199074    150   64614938        C
2199075    150   64716982        C
2199076    150   65355704        C

[2199077 rows x 3 columns]
```

And now use that to compute the number of documents at each work level:

```
qwork = qpw.groupby(['id', 'quality'])['page_id'].count()
qwork
```

```
id    quality
1     Stub        1527
      Start       2822
      C           1603
      B            610
```

35

```
    GA           240
                 ...
150  Start       138
     C           127
     B            35
     GA           16
     FA            8
Name: page_id, Length: 636, dtype: int64
```

Now we need to convert this into target exposure levels. This function will, given a series of counts for each work level, compute the expected exposure a page at that work level should receive.

```python
def qw_tgt_exposure(qw_counts: pd.Series) -> pd.Series:
    if 'id' == qw_counts.index.names[0]:
        qw_counts = qw_counts.reset_index(level='id', drop=True)
    qwc = qw_counts.reindex(work_order, fill_value=0).astype('i4')
    tot = int(qwc.sum())
    da = metrics.discount(tot)
    qwp = qwc.shift(1, fill_value=0)
    qwc_s = qwc.cumsum()
    qwp_s = qwp.cumsum()
    res = pd.Series(
        [np.mean(da[s:e]) for (s, e) in zip(qwp_s, qwc_s)],
        index=qwc.index
    )
    return res
```

We'll then apply this to each topic, to determine the per-topic target exposures:

```python
qw_pp_target = qwork.groupby('id').apply(qw_tgt_exposure)
qw_pp_target.name = 'tgt_exposure'
qw_pp_target
```

```
C:\Users\michaelekstrand\Miniconda3\envs\wptrec\lib\site-packages\numpy\core\fromnumeric.py:3440: Runtin
  return _methods._mean(a, axis=axis, dtype=dtype,
C:\Users\michaelekstrand\Miniconda3\envs\wptrec\lib\site-packages\numpy\core\_methods.py:189: RuntimeWa
  ret = ret.dtype.type(ret / rcount)

id    quality
1     Stub       0.114738
      Start      0.087373
      C          0.081146
      B          0.079298
      GA         0.078702
                    ...
150   Start      0.154202
      C          0.127359
      B          0.120441
      GA         0.118827
      FA         0.118126
Name: tgt_exposure, Length: 636, dtype: float32
```

We can now merge the relevant document work categories with this exposure, to compute the target exposure for each relevant document:

```
qp_exp = qpw.join(qw_pp_target, on=['id', 'quality'])
qp_exp = qp_exp.set_index(['id', 'page_id'])['tgt_exposure']
qp_exp.index.names = ['q_id', 'page_id']
qp_exp
```

```
q_id  page_id
1     572           0.081146
      627           0.078438
      903           0.081146
      1193          0.079298
      1542          0.078702
                         ...
150   63656179      0.154202
      63807245           NaN
      64614938      0.127359
      64716982      0.127359
      65355704      0.127359
Name: tgt_exposure, Length: 2199077, dtype: float32
```

A.6.2 Geographic Alignment

Now that we've computed per-page target exposure, we're ready to set up the geographic alignment vectors for computing the per-*group* expected exposure with geographic data.

We're going to start by getting the alignments for relevant documents for each topic:

```
qp_geo_align = qrels.join(page_geo_align, on='page_id').set_index(['id', 'page_id'])
qp_geo_align.index.names = ['q_id', 'page_id']
qp_geo_align
```

```
              Unknown  Africa  Antarctica  Asia  Europe  \
q_id page_id
1    572          1.0     0.0         0.0   0.0     0.0
     627          1.0     0.0         0.0   0.0     0.0
     903          1.0     0.0         0.0   0.0     0.0
     1193         1.0     0.0         0.0   0.0     0.0
     1542         1.0     0.0         0.0   0.0     0.0
...                ...     ...         ...   ...     ...
150  63656179     1.0     0.0         0.0   0.0     0.0
     63807245     NaN     NaN         NaN   NaN     NaN
     64614938     1.0     0.0         0.0   0.0     0.0
     64716982     1.0     0.0         0.0   0.0     0.0
     65355704     1.0     0.0         0.0   0.0     0.0

              Latin America and the Caribbean  Northern America  Oceania
q_id page_id
1    572                                  0.0               0.0      0.0
     627                                  0.0               0.0      0.0
     903                                  0.0               0.0      0.0
     1193                                 0.0               0.0      0.0
     1542                                 0.0               0.0      0.0
...                                       ...               ...      ...
150  63656179                             0.0               0.0      0.0
     63807245                             NaN               NaN      NaN
```

```
    64614938                                  0.0              0.0       0.0
    64716982                                  0.0              0.0       0.0
    65355704                                  0.0              0.0       0.0
```

`[2199077 rows x 8 columns]`

Now we need to compute the per-query target exposures. This starst with aligning our vectors:

`qp␣geo␣exp, qp␣geo␣align = qp␣exp.align(qp␣geo␣align, fill␣value=0)`

And now we can multiply the exposure vector by the alignment vector, and summing by topic - this is equivalent to the matrix-vector multiplication on a topic-by-topic basis.

```
qp␣aexp = qp␣geo␣align.multiply(qp␣geo␣exp, axis=0)
q␣geo␣align = qp␣aexp.groupby('q␣id').sum()
```

Now things get a *little* weird. We want to average the empirical distribution with the world population to compute our fairness target. However, we don't have empirical data on the distribution of articles that do or do not have geographic alignments.

Therefore, we are going to average only the *known-geography* vector with the world population. This proceeds in N steps:

1. Normalize the known-geography matrix so its rows sum to 1.
2. Average each row with the world population.
3. De-normalize the known-geography matrix so it is in the original scale, but adjusted w/ world population
4. Normalize the *entire* matrix so its rows sum to 1

Let's go.

`qg␣known = q␣geo␣align.drop(columns=['Unknown'])`

Normalize (adding a small value to avoid division by zero - affected entries will have a zero numerator anyway):

```
qg␣ksums = qg␣known.sum(axis=1)
qg␣kd = qg␣known.divide(np.maximum(qg␣ksums, 1.0e-6), axis=0)
```

Average:

`qg␣kd = (qg␣kd + world␣pop) * 0.5`

De-normalize:

`qg␣known = qg␣kd.multiply(qg␣ksums, axis=0)`

Recombine with the Unknown column:

`q␣geo␣tgt = q␣geo␣align[['Unknown']].join(qg␣known)`

Normalize targets:

```
q␣geo␣tgt = q␣geo␣tgt.divide(q␣geo␣tgt.sum(axis=1), axis=0)
q␣geo␣tgt
```

```
        Unknown      Africa     Antarctica        Asia      Europe  \
q_id
1       0.575338    0.043635   3.278897e-08   0.153851    0.098450
2       0.173889    0.069608   6.378567e-08   0.294269    0.280798
3       0.234897    0.101882   5.907510e-08   0.278161    0.215027
4       0.312664    0.076008   8.262075e-05   0.246140    0.145192
5       0.182143    0.063760   6.314834e-08   0.273795    0.046710
...          ...         ...            ...        ...         ...
146     0.292441    0.090378   5.463208e-08   0.299627    0.067556
147     0.434276    0.060053   4.368069e-08   0.195520    0.130625
148     0.637050    0.033542   2.802409e-08   0.233693    0.045680
149     0.370828    0.061724   4.857964e-08   0.243518    0.172170
150     0.414091    0.062031   4.523918e-08   0.208319    0.131270

        Latin America and the Caribbean   Northern America    Oceania
q_id
1                              0.025042           0.065388   0.038296
2                              0.046323           0.115193   0.019920
3                              0.071196           0.077784   0.021053
4                              0.058319           0.143947   0.017648
5                              0.061549           0.366345   0.005697
...                                 ...                ...        ...
146                            0.045686           0.178497   0.025815
147                            0.061604           0.091005   0.026916
148                            0.018613           0.025322   0.006099
149                            0.040886           0.073876   0.036999
150                            0.042203           0.116868   0.025218

[106 rows x 8 columns]
```

This is our group exposure target distributions for each query, for the geographic data. We're now ready to set up the matrix.

```
train_geo_qtgt = q_geo_tgt.loc[train_topics['id']]
eval_geo_qtgt = q_geo_tgt.loc[eval_topics['id']]

t2_train_geo_metric = metrics.Task2Metric(train_qrels.set_index('id'),
                                          page_geo_align, page_work,
                                          train_geo_qtgt)
binpickle.dump(t2_train_geo_metric, 'task2-train-geo-metric.bpk', codec=codec)

INFO:binpickle.write:pickled 2018 bytes with 9 buffers

t2_eval_geo_metric = metrics.Task2Metric(eval_qrels.set_index('id'),
                                         page_geo_align, page_work,
                                         eval_geo_qtgt)
binpickle.dump(t2_eval_geo_metric, 'task2-eval-geo-metric.bpk', codec=codec)

INFO:binpickle.write:pickled 2014 bytes with 9 buffers
```

A.6.3   Intersectional Alignment

Now we need to compute the intersectional targets for Task 2. We're going to take a slightly different approach here, based on the intersectional logic for Task 1, because we've come up with better ways to write the code, but the effect is the same: only known aspects are averaged.

We'll write a function very similar to the one for Task 1:

```python
def query_xideal(qdf, ravel=True):
    pages = qdf['page_id']
    pages = pages[pages.isin(page_xalign.indexes['page'])]
    q_xa = page_xalign.loc[pages.values, :, :]

    # now we need to get the exposure for the pages, and multiply
    p_exp = qp_exp.loc[qdf.name]
    assert p_exp.index.is_unique
    p_exp = xr.DataArray(p_exp, dims=['page'])

    # and we multiply!
    q_xa = q_xa * p_exp

    # normalize into a matrix (this time we don't clear)
    q_am = q_xa.sum(axis=0)
    q_am = q_am / q_am.sum()

    # compute fractions in each section - combined with q_am[0,0], this should be about 1
    q_fk_all = q_am[1:, 1:].sum()
    q_fk_geo = q_am[1:, :1].sum()
    q_fk_gen = q_am[:1, 1:].sum()

    # known average
    q_am[1:, 1:] *= 0.5
    q_am[1:, 1:] += int_tgt * 0.5 * q_fk_all

    # known-geo average
    q_am[1:, :1] *= 0.5
    q_am[1:, :1] += geo_tgt_xa * 0.5 * q_fk_geo

    # known-gender average
    q_am[:1, 1:] *= 0.5
    q_am[:1, 1:] += gender_tgt_xa * 0.5 * q_fk_gen

    # and return the result
    if ravel:
        return pd.Series(q_am.values.ravel())
    else:
        return q_am
```

Test this function out:

```python
query_xideal(qdf, ravel=False)
```

```
<xarray.DataArray (geography: 8, gender: 4)>
array([[5.40211229e-01, 1.22904624e-02, 2.26610467e-02, 1.75635724e-04],
       [3.80909493e-02, 2.90804953e-03, 2.59344827e-03, 4.25392005e-05],
       [2.85527900e-08, 2.09691080e-09, 2.09691080e-09, 4.23618344e-11],
       [1.34695670e-01, 8.88355123e-03, 1.01072347e-02, 1.64648516e-04],
       [8.71895859e-02, 2.97387866e-03, 8.25814408e-03, 2.84372324e-05],
       [2.16878846e-02, 1.67819032e-03, 1.65196427e-03, 2.36185304e-05],
```

```
           [5.32652519e-02, 2.51534798e-03, 9.59370956e-03, 1.36109402e-05],
           [3.48679417e-02, 4.71346052e-04, 2.95512391e-03, 1.46710935e-06]])
Coordinates:
  * geography  (geography) object 'Unknown' 'Africa' ... 'Oceania'
  * gender     (gender) object 'unknown' 'female' 'male' 'third'
```

And let's go!

```
q_xtgt = qrels.groupby('id').progress_apply(query_xideal)
q_xtgt
```

{"model_id":"","version_major":2,"version_minor":0}

```
            0         1         2         3         4         5         6   \
id
1     0.540211  0.012290  0.022661  0.000176  0.038091  0.002908  0.002593
2     0.135109  0.010633  0.027958  0.000201  0.063400  0.003032  0.003115
3     0.185923  0.018891  0.029817  0.000245  0.018607  0.033486  0.049321
4     0.283620  0.008665  0.020234  0.000145  0.069568  0.003021  0.003361
5     0.102865  0.021347  0.057531  0.000396  0.017768  0.022647  0.022888
..         ...       ...       ...       ...       ...       ...       ...
146   0.242344  0.017108  0.032738  0.000250  0.033631  0.031692  0.024843
147   0.380085  0.025582  0.026067  0.001304  0.028472  0.017849  0.014999
148   0.620663  0.005550  0.010755  0.000082  0.031143  0.001188  0.001188
149   0.365415  0.002870  0.002516  0.000027  0.060143  0.000783  0.000783
150   0.228180  0.057917  0.127065  0.000930  0.014522  0.021052  0.026136

            7             8             9   ...        22        23        24   \
id                                          ...
1     0.000043  2.855279e-08  2.096911e-09  ...  0.001652  0.000024  0.053265
2     0.000059  5.789347e-08  2.916166e-09  ...  0.002811  0.000033  0.099662
3     0.000471  1.300894e-08  2.280358e-08  ...  0.032680  0.000257  0.019061
4     0.000059  8.261490e-05  2.894759e-09  ...  0.002071  0.000033  0.127457
5     0.000458  1.758741e-08  2.255280e-08  ...  0.034300  0.000254  0.104245
..         ...           ...           ...  ...       ...       ...       ...
146   0.000212  3.349054e-08  1.046506e-08  ...  0.009259  0.000118  0.164611
147   0.000207  2.317530e-08  1.019747e-08  ...  0.018809  0.000115  0.050914
148   0.000024  2.563480e-08  1.182699e-09  ...  0.000659  0.000013  0.020264
149   0.000016  4.700522e-08  7.793387e-10  ...  0.002786  0.000009  0.071839
150   0.000320  1.332948e-08  1.579530e-08  ...  0.016691  0.000178  0.040721

            25        26        27        28        29        30            31
id
1     0.002515  0.009594  0.000014  0.034868  0.000471  0.002955  1.467109e-06
2     0.002826  0.012684  0.000019  0.017811  0.000468  0.001639  2.040304e-06
3     0.018746  0.039812  0.000164  0.004908  0.005947  0.010179  1.595459e-05
4     0.002601  0.013870  0.000019  0.015398  0.000279  0.001968  2.025326e-06
5     0.012692  0.249255  0.000146  0.002342  0.000883  0.002456  1.577913e-05
..         ...       ...       ...       ...       ...       ...           ...
146   0.010455  0.003362  0.000068  0.013121  0.012324  0.000362  7.321910e-06
147   0.022379  0.017457  0.000066  0.016122  0.005498  0.005220  7.134685e-06
148   0.000380  0.004670  0.000008  0.003416  0.000041  0.002642  8.274784e-07
149   0.000250  0.001781  0.000005  0.036944  0.000027  0.000027  5.452665e-07
```

```
150   0.017971   0.058074   0.000103   0.009900   0.000547   0.013698   1.071915e-03

[106 rows x 32 columns]

train_qtgt = q_xtgt.loc[train_topics['id']]
eval_qtgt = q_xtgt.loc[eval_topics['id']]

t2_train_metric = metrics.Task2Metric(train_qrels.set_index('id'),
                                      page_xa_df, page_work,
                                      train_qtgt)
binpickle.dump(t2_train_metric, 'task2-train-metric.bpk', codec=codec)

INFO:binpickle.write:pickled 1879 bytes with 9 buffers

t2_eval_metric = metrics.Task2Metric(eval_qrels.set_index('id'),
                                     page_xa_df, page_work,
                                     eval_qtgt)
binpickle.dump(t2_eval_metric, 'task2-eval-metric.bpk', codec=codec)

INFO:binpickle.write:pickled 1875 bytes with 9 buffers
```