Litescale: A Lightweight Tool for Best-worst Scaling Annotation

Valerio Basile
University of Turin
valerio.basile@unito.it

Christian Cagnazzo
University of Turin
christian.cagnazzo@edu.unito.it

Abstract

Best-worst Scaling (BWS) is a methodology for annotation based on comparing and ranking instances, rather than classifying or scoring individual instances. Studies have shown the efficacy of this methodology applied to NLP tasks in terms of a higher quality of the datasets produced by following it. In this system demonstration paper, we present LITESCALE, a free software library to create and manage BWS annotation tasks. LITESCALE computes the tuples to annotate, manages the users and the annotation process, and creates the final gold standard. The functionalities of LITESCALE can be accessed programmatically through a Python module, or via two alternative user interfaces, a textual console-based one and a graphical Web-based one. We further developed and deployed a fully online version of LITESCALE complete with multi-user support.

1 Introduction

Annotation is a cornerstone of Computational Linguistics and Natural Language Processing (NLP). Much of modern NLP is based on machine learning, and in particular on supervised machine learning. As a consequence, large amounts of manually annotated data are constantly in high demand, and the quality of the annotation directly influence the predictive capability of models trained on them.

The annotation of natural language resources comes in many forms of varying complexity and levels of abstraction. However, it is possible to categorize most of the approaches in three main families. *Categorial* annotation is perhaps the most common approach, whereas each instance of a dataset is associated with a label from a fixed set of options. Annotation can also consist of *scalar* values, that is, numeric values on a predetermined scale. Finally, *ranking* is a type of annotation where multiple instances are put in a certain order by the annotator, who therefore does not make judgments on instances in isolation but rather on groups of them.

Recent literature highlights the advantages of the ranking strategy, in terms of the quality of the annotation produced with it, in particular when dealing with subjective aspects of natural language (Yannakakis et al., 2018). The Best-Worst Scaling model (BWS) is a ranking-based annotation process developed by Louviere et al. (2015). BWS has been proved to be beneficial to the quality of the resulting gold standard data for subjective-related tasks such as emotion detection (Kiritchenko and Mohammad, 2017) and hate speech detection (Poletto et al., 2019). The recent work of De Bruyne, Luna and De Clercq, Orphée and Hoste, Veronique (2021) further proved the validity of BWS as a method of annotation for sentiment-related tasks, dominance in particular.

Several general-purpose annotation tools have been proposed in the literature. Among the most popular, WebAnno (Eckart de Castilho et al., 2016) and Brat (Stenetorp et al., 2012) both provide Web-based interfaces and the possibility to implement rich annotation schemas, including annotation at the word and span level, and links between annotations. A number of online services are also available to perform linguistic annotation, especially in a crowdsourced fashion, such as Amazon Mechanical Turk[1], Appen[2], or LightTag[3]. All these systems, however, allow the user to devise schemas that associate labels or scores to individual units of text, whether words, spans, sentences or others. In other words, they natively give the possibility of implementing categorial or ranking annotations. To our knowledge, there is no publicly available system

[1] https://www.mturk.com/
[2] https://appen.com/
[3] https://www.lighttag.io/

Proceedings of Recent Advances in Natural Language Processing, pages 121–127
Sep 1–3, 2021.
https://doi.org/10.26615/978-954-452-072-4_015

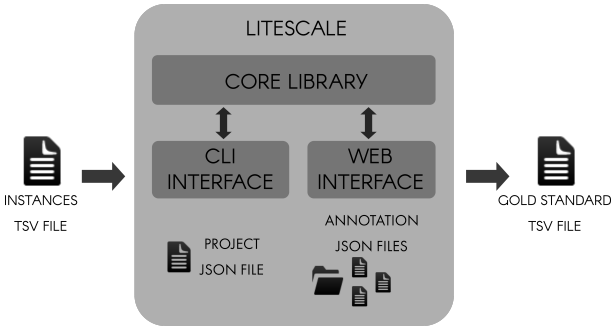

Figure 1: Software architecture of LITESCALE.

that implements a general-purpose ranking annotation methodology, in particular of the Best-worst Scaling flavour.

2 Best-worst Scaling Annotation

In Best-worst Scaling (BWS) annotation, the annotator is shown a tuple consisting of a fixed number of instances, and a phenomenon to annotate. The annotator is then asked to select the instance from the tuple which expresses the phenomenon to the maximum extent, and the one that expresses it to the least extent. For example, in a sentiment polarity annotation task, the annotator may be asked to select the sentences conveying the most positive and the most negative sentiment.

Starting from the set of instances to annotate, tuples must be created such that:

- The instances appears in tuples in a random order.
- Each instance appears in a predetermined number of tuples.
- The same pair of instances never appears in the same tuple more than once.

The size of the tuples and the number of tuples in which each instance appears are parameters to be determined at the time of the creation of the tuples. The ratio between the values of these two parameters determines the number of tuples with respect to the number of input instances. In particular, if the two parameters are equal, then the number of tuples will be approximately the same as the instances to annotate.

Since the number of tuples where any individual instance appears equals the size of the tuples (four items), the number of final questions for the annotators is roughly equivalent to the number of items to be annotated.

Once the annotation is complete, a score is computed for each instance as the difference between the number of times the instance was selected as "best" and the number of times it was selected as "worst". The score can easily be normalized to fit a $[-1, 1]$ or $[0, 1]$ interval, depending on the needs of the downstream application.

Since the tuples are computed once, at the beginning of the annotation process, every annotator is presented the same set of instances. It is therefore possible to compute standard measures of inter-annotator agreement such as Fleiss' Kappa or Krippendorf's Alpha (Artstein and Poesio, 2008).

3 Litescale

In this section, we introduce LITESCALE, a free and open source software package that implements the Best-worst Scaling annotation framework. LITESCALE is composed of a core Python library implementing the core functions, and two alternative interfaces, a console-based textual one and a Web-based graphical one. The user can create annotation projects by providing a tab-separated text file, and obtain a similarly formatted file at the end of the annotation. A diagram depicting the information flow and the modules of LITESCALE is in Figure 1.

3.1 Core Functions

The core library of LITESCALE implements a series of functions to create and manage BWS annotation tasks, called *projects*. A project is created starting from a simple text file, where each line is a tab-separated pair of an instance identifier (which can be any string) and its textual content. Upon the creation of a new project, a JSON file is created representing all the information relative to the project.

Tuple creation. The function to create a new project takes five inputs: a name, the label for the phenomenon to annotate, the file containing the instances, and two parameters for the creation of the tuples from the set of input instances. Such parameters are the size of the tuples to create (*tuple size)* and the number of tuples in which a single instance will occur (*instance replication*). Given these two parameters, the system implements the three constraints described in Section 1 by leveraging Gauss' modular arithmetic. More precisely, calling s the tuple size and r the replication factor, and n the

total number of instances, tuple are composed of instances with indexes in the form:

$$(xs^{j+1} + is^j) \bmod n$$
$$0 \leq x < \lfloor n/s \rfloor$$
$$0 \leq i < s$$
$$0 \leq j < r$$

This formula ensures that the BWS conditions are met, by "wrapping up" the indexes in an appropriate way. For instance, with $s = 4$, $r = 4$, and $n = 101$, the first tuples are identified by the following indexes: $(1, 2, 3, 4)$, $(5, 6, 7, 8)$, ... However, once the indexes exceed n, the modular arithmetic recombines them: $(1, 5, 9, 13)$, ..., $(81, 85, 89, 93)$, $(97, 101, 4, 8)$, and so on.

While this method ensures to fully include all the input instances into the tuples, due to the inner working of the modular arithmetic the formula works correctly under the condition that n (the number of instances) and s (the size of the tuples) are be *co-primes*, i.e., they need not have common divisors other than 1. As a workaround, the LITESCALE library automatically exclude a number up to $s - 1$ of instances from the process, to prevent the formula to run into a corner case. Furthermore, for some combinations of parameters it is mathematically impossible to have all the instances appear in exactly r tuples. More precisely, there will be up to $s - 1$ instances that occur $r - 1$ times, i.e., a considerably small set of instances will lack exactly one annotation at the end of the process.

BWS annotation. The library creates a directory to store all the annotations for a project as a series of JSON files, one for each user partitipating to the annotation task. The *next_tuple* function checks the current status of the annotation and returns the first non-annotated tuple to be presented to the user. The *annotate* function takes a pair of tuple indexes indicating the "best" and the "worst" and updates the annotation JSON correspondingly. The *progress* function returns the number of currently annotated tuples by a given user and the total number of tuples in the project, in order to compute the advancement status of the project as a percentage or progress bar.

Gold standard. The other key function in the LITESCALE core library is the one that computes the gold standard. This function collects all the annotations provided by the users for a specific project, and computes the gold standard according to the BWS methodology. For each instance, library counts the number of times it was judged "best" and the number of times it was judges "worst", and computes the difference between these numbers. The result is then normalized using the minimum and maximum values of the differences across all instances, in order to return a score between 0 and 1, indicating the relevance of the phenomenon object of the annotation project in each instance. The gold standard dataset is finally written out to a text tab-separated file consisting of three columns, namely instance ID, text, and score.

3.2 Command-line Interface

The command-line interface (CLI) of LITESCALE is implemented with the *PyInquirer*[4] Python library. It makes use of the functions provided by the LITESCALE core library and exposes an interactive, menu-based interface to easily navigate through its functionalities.

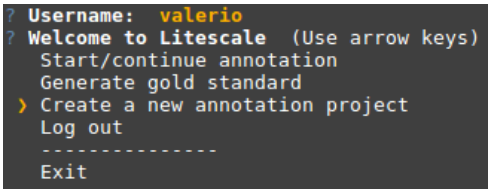

Figure 2: LITESCALE command-line interface: login and start menu.

Upon starting, the CLI asks for a username to log in, and keeps memory of the last user that had logged in previous sessions. No password is requested, since the application is intended to be for single user interaction in individual installations. After logging in, a menu with a few options is proposed to the user, as shown in Figure 2.

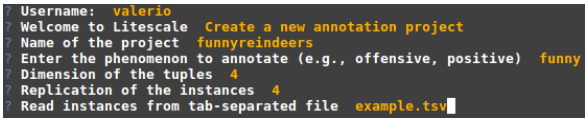

Figure 3: LITESCALE Command-line interface: creation of a new annotation project.

- *Start/continue annotation*: lists the available annotation projects, and start the annotation of the selected project.
- *Generate gold standard*: lists the available annotation projects, and creates the gold stan-

[4] https://github.com/CITGuru/PyInquirer

dard dataset from all the annotations collected for the selected project.

- *Create a new annotation project*: prompts a sub-menu wizard to create a new project, as shown in Figure 3.
- *Log out*: returns to the login prompt.
- *Exit*: quits the application.

The creation of a new annotation project involves a small set of questions for the user. These include a name for the project, a label for the phenomenon to annotate, values for the two parameters for the creation of the tuples (tuple size and instance replication), and the path to a tab-separated file from which the instances will be read.

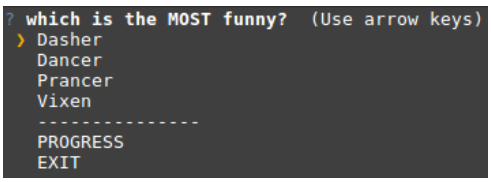

Figure 4: LITESCALE Command-line interface: best-worst scaling annotation.

Selecting *start/continue annotation* from the main menu, the user can choose one of the available projects and start the BWS annotation. The annotation automatically starts from the beginning, if this option is selected for the first time, or it continues at the point where it was left by the logged in annotator for the selected project.

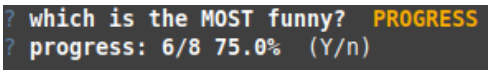

Figure 5: LITESCALE Command-line interface: checking the progress.

For the BWS annotation, the CLI shows two questions in sequence to the user, in the form "which is the MOST *label*?" and "which is the LEAST *label*?", followed by the list of the instances in a tuple, where *label* is the label of the phenomenon for the selected project. An example of the annotation interface is shown in Figure 4. At any moment, the user can select *PROGRESS* to check how many instances are left to complete the task (Figure 5), or *EXIT* to return to the main menu.

3.3 Web-based Interface

In order to provide a more versatile user experience, LITESCALE is equipped with a Web-based interface, as an alternative to the console-base one described in the previous section. A Web application was developed with the Bottle Python library[5] that incorporates the core library of LITESCALE and exposes a local HTTP server.

Figure 6: LITESCALE Web-based interface: main menu.

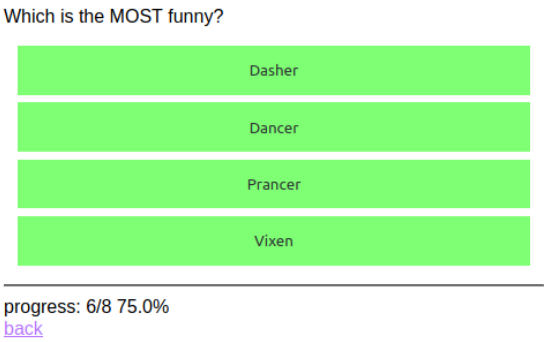

Figure 7: LITESCALE Web-based interface: BWS annotation.

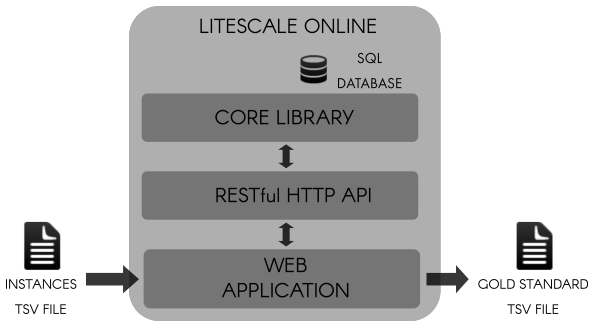

Figure 8: Software architecture of LITESCALE online.

Executing the server script and pointing a Web browser to the indicated URL, the Web interface

[5] https://bottlepy.org/

of LITESCALE is shown, initially prompting for a username, exactly as its CLI counterpart. Once the user has logged in, the main menu is displayed as in Figure 6.

The Web-based interface provides the same functionalities as the CLI, and the annotation workflow is identical (Figure 7). Furthermore, the two interfaces are by default part of the same installation of LITESCALE and make use of the same core library. Therefore, an annotation task can start with one interface and switch seamlessly to the other and back at any time. The annotations created with the two interfaces are exactly in the same format, and can therefore be merged without any extra work.

4 Multi-user Online Platform

Despite being a standalone application, LITESCALE is natively multi-user, because the annotations produces by different users are represented as different JSON files. These files can reside on the same machine, or different machines if multiple installation of LITESCALE took place. In the latter case, it is sufficient to copy the annotation files of a project from one installation to another to generate a gold standard comprising all the annotations. This process, however can be tedious and error-prone. Moreover, the version of LITESCALE presented so far needs to be executed as a series of Python scripts, therefore requiring at least a minimal amount of skill in such technology from the annotators. To overcome these issues, and provide an even easier and more accessible user experience, we developed a fully online, multi-user version of LITESCALE, described in this section.

The online version of LITESCALE offers the same functionalities as the standalone version described in the previous section, with a user interface very similar to the Web-based interface in Section 3.3. The front-end was however implemented from scratch, while retaining the core library for the basic functions, although with some modifications. LITESCALE online is a Web application implemented with the *Flask*[6] Python framework. More precisely, the software is designed in a modular way. A RESTful HTTP API is provided in order to expose the core library functionalities, and a Web application provides the user interface by connecting to the API. Figure 8 shows the modular architecture of this version of LITESCALE.

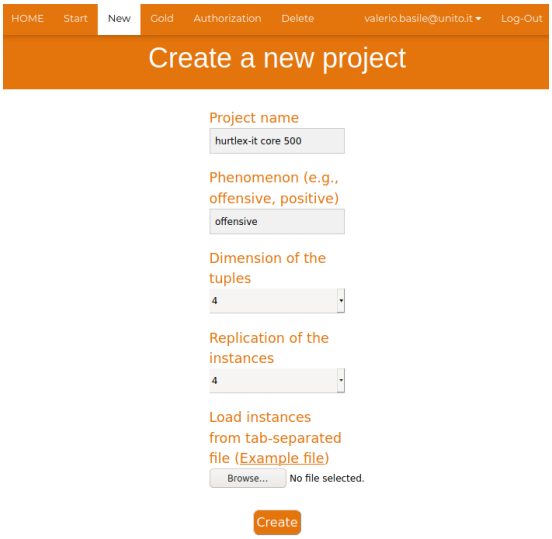

Figure 9: LITESCALE online: creating a new annotation project.

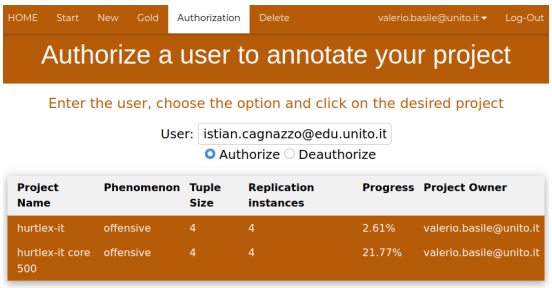

Figure 10: LITESCALE online: management of user authorization on projects.

The RESTful API is intended as a layer of abstraction to facilitate the future development of tools that access the core library functions remotely, or application that provide alternative interfaces, such as mobile applications. The RESTful API is implemented with the *Flask-RESTful* extension of the Flask framework[7]. The HTTP verbs and their semantics are summarized in Table 1.

In order to make LITESCALE online ready to scale up the number of users and projects, the core library has been modified. In particular, the inner representation and persistence of instances, projects, tuples, and annotations, does not rely on JSON files but rather on a SQL database.

The workflow in the online version of LITESCALE is substantially the same as the standalone version, including the interface to create new projects (Figure 9) and the annotation interface itself. However, additional functionalities were implemented to account for the online multi-user envi-

[6] https://flask.palletsprojects.com

[7] https://flask-restful.readthedocs.io

Method	Endpoint	Description
POST	*/users*	Creates a user
DELETE	*/users*	Deletes a user
GET	*/projectList*	Retrieve the list of projects of a user
GET	*/projects*	Retrieves the properties of a project
POST	*/projects*	Creates a project
DELETE	*/projects*	Deletes a project
GET	*/tuples*	Retrieves the tuples of a project
POST	*/annotations*	Creates a new annotation
GET	*/gold*	Generates the gold standard
GET	*/progress*	Returns the current progress of an annotation task
POST	*/authorizations*	Adds a user to a project
DELETE	*/authorizations*	Removes a user from a project

Table 1: The LITESCALE RESTful API.

ronment. The users can sign up with a valid email address and set up proper log in credentials after receiving a confirmation by email. The user authentication features are managed by the *Werkzeug*[8] Python library, including the storage of encrypted passwords in the database. Moreover, the association of users to projects is also managed through the Web application. By default, when a project is created, the user who created it assumes the role of *owner* of that project. The owner of a project can invite other users to join their projects, by indicating a valid email address (Figure 10). The recipients of such invitation will receive an automated email notification, inviting them to sign up to LITESCALE if they are not already registered.

5 Evaluation

We conducted a pilot test in order to evaluate the efficacy of Litescale in supporting the Best-worst Scaling annotation of datasets for NLP tasks. We extracted a sample of short hotel reviews in English from the list of 1,000 hotels and their reviews provided by Datafiniti's Business Database[9]. We selected the 20 shortest reviews for each of the five ratings (1 to 5 stars) for a total of 100 instances, shuffled then randomly and created an annotation task in Litescale. Three annotators were asked to perform the annotation and record the times spent annotating each 10 annotations (pairs of best/worst judgments). We computed the Pearson correlation between the scores resulting from Litescale and the original ratings associated to the reviews, obtaining a score of 0.77, indicating strong correlation with the ground truth. The agreement between the annotators is relatively high, considering the subjectivity of the sentiment polarity annotation task. Interestingly, the three annotators agreed more on the *worst* judgments (two annotators agreed 91% of the times, all three agreed 66% of the times) than on the *best* ones (two annotators agreed 88% of the times, all three agreed 58% of the times).

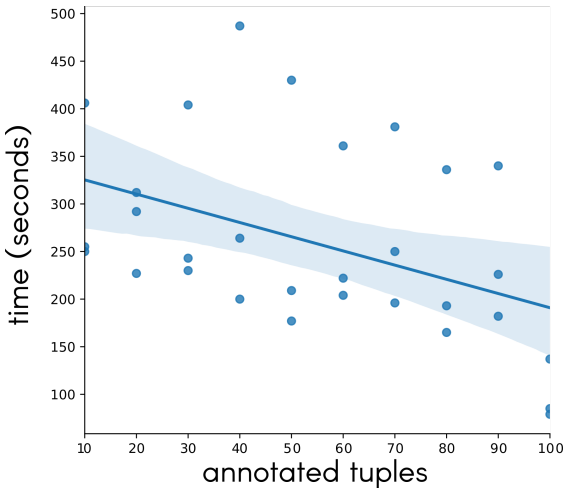

Figure 11: Recorded annotation times during the LITESCALE evaluation experiment. Each 10 tuples, the three annotators recorded the time spent on the annotation, shown as a dot in the plot.

The annotation took about 43 minutes on average. However, the average time spent annotating tends to decrease consistently as the annotation task progresses, as shown in Figure 11. This is not surprising, considering the nature of the BWS annotation, where the same instance is shown multiple times, and therefore the annotator gains familiarity with the instances over time.

6 Conclusion and Future Work

We introduced LITESCALE, a flexible software tool to create and manage linguistic annotation tasks based on the Best-worst scaling methodology. LITESCALE is easy to use, free and open source[10], and ships with different user interfaces. Moreover, a multi-user online version of LITESCALE is also presented[11], which runs entirely in a Web browser

[8] https://werkzeug.palletsprojects.com

[9] https://www.kaggle.com/datafiniti/hotel-reviews?select=Datafiniti_Hotel_Reviews_Jun19.csv

[10] The source code of LITESCALE is available on the Github repository https://github.com/valeriobasile/litescale

[11] LITESCALE online is available at http://lite-env.eba-jhijbmtj.eu-west-3.elasticbeanstalk.com/home

and provides all the functionalities of the original software.

While the BWS methodology has been thoroughly tested in the literature, this particular tool has not been systematically tested for its usability (but it is being used at the moment for the creation of several language resources). Although all the main features are implemented, there is always room to add extra functionalities and improving the existing ones.

A short video describing the main functions of LITESCALE is abailable on YouTube: https://youtu.be/SozWDMH2ah0

Acknowledgments

This work is partially funded by the project "Be Positive!" (under the 2019 "Google.org Impact Challenge on Safety" call).

References

Ron Artstein and Massimo Poesio. 2008. Inter-Coder Agreement for Computational Linguistics. *Computational Linguistics*, 34(4):555–596.

Richard Eckart de Castilho, Éva Mújdricza-Maydt, Seid Muhie Yimam, Silvana Hartmann, Iryna Gurevych, Anette Frank, and Chris Biemann. 2016. A web-based tool for the integrated annotation of semantic and syntactic structures. In *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH)*, pages 76–84, Osaka, Japan. The COLING 2016 Organizing Committee.

De Bruyne, Luna and De Clercq, Orphée and Hoste, Veronique. 2021. Annotating affective dimensions in user-generated content : comparing the reliability of best-worst scaling, pairwise comparison and rating scales for annotating valence, arousal and dominance. *LANGUAGE RESOURCES AND EVALUATION*.

Svetlana Kiritchenko and Saif Mohammad. 2017. Best-worst scaling more reliable than rating scales: A case study on sentiment intensity annotation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 465–470. ACL.

Jordan J Louviere, Terry N Flynn, and Anthony Alfred John Marley. 2015. *Best-worst scaling: Theory, methods and applications*. Cambridge University Press.

Fabio Poletto, Valerio Basile, Cristina Bosco, Viviana Patti, and Marco Stranisci. 2019. Annotating hate speech: Three schemes at comparison. In *6th Italian Conference on Computational Linguistics, CLiC-it 2019*, volume 2481, pages 1–8. CEUR-WS.

Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. brat: a web-based tool for NLP-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107, Avignon, France. Association for Computational Linguistics.

Georgios Yannakakis, Roddy Cowie, and Carlos Busso. 2018. The ordinal nature of emotions: An emerging approach. *IEEE Transactions on Affective Computing*, pages 1–20. Early Access.