C-Test Collector: A Proficiency Testing Application to Collect Training Data for C-Tests

Christian Haring, René Lehmann, Andrea Horbach and Torsten Zesch
Language Technology Lab, University Duisburg-Essen
(christian.haring|rene.lehmann)@stud.uni-due.de
(andrea.horbach|torsten.zesch)@uni-due.de

Abstract

We present the C-Test Collector, a web-based tool that allows language learners to test their proficiency level using c-tests. Our tool collects anonymized data on test performance, which allows teachers to gain insights into common error patterns. At the same time, it allows NLP researchers to collect training data in order to be able to generate c-test variants at the desired difficulty level.

1 Introduction

The c-test (Raatz and Klein-Braley, 1981) is a reduced redundancy exercise frequently used for language proficiency testing. In a c-test, the second half of every second word is replaced by a gap, and the task of the test-taker is to reconstruct the text. For example, the word *redundancy* would be replaced with *redun___*.

C-tests are known to correlate well with general language proficiency (Grotjahn, 2002) and are fast to take and to evaluate. Thus, they are even used in commercial systems such as onDaF (Grotjahn, 2010).

Despite these advantages, the creation of a well-working c-test is a time-consuming task even for language experts. One reason is that the gap scheme is influenced e.g. by Named Entities or other words that would be hard to guess for learners. Improved tool support for curating c-tests has mitigated this problem to some extent (Zesch et al., 2018).

Even with technological support, it is still hard (even for experts) to judge the difficulty of a given test without running a pilot study (Beinborn et al., 2014). Thus, there has been research on predicting (Beinborn et al., 2014) or manipulating (Lee et al., 2019) the difficulty of c-tests. These approaches heavily rely on the availability of training data, i.e. c-tests which have been taken by enough learners so that the solution probability for individual gaps can be reliably estimated. Influencing factors on the part of the student, such as the native language, make the difficulty prediction even more dependent on the availability of the right kind of training data.

In this paper, we present a web-application that makes it easy to collect such training data, by providing an easy-to-use platform for taking a c-test online. In the remainder of this paper, we describe the two ways to interact with the system: (i) as a user taking c-tests and receiving a proficiency assessment, and (ii) as an administrator uploading new tests and accessing the aggregated test results of the users.

2 User Perspective

Figure 1 illustrates the interaction steps of a user with the application.

Self Assessment To allow for an easy start with the tool and to avoid data security issues, we do not create user profiles, but ask users only for the language they want to learn and a self-assessment into one out of six proficiency levels, roughly meant to correspond to the six CEFR levels (Council of Europe, 2009). Optionally, a user can provide age and native language. This meta-information is stored together with the solutions provided by the user for a particular test.

Test Taking After selecting a language and proficiency level, the user is delivered a suitable c-test (see step 2 in Figure 1). Figure 2 gives an example of how a test looks for the user. Tests can be skipped and revisited again at a later stage. If the user has finished a test they are redirected to an evaluation page.

A user may always update their language settings to receive a new test (left-hand side of the upper box). On the right hand side of the upper box, the

Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications, pages 180–184

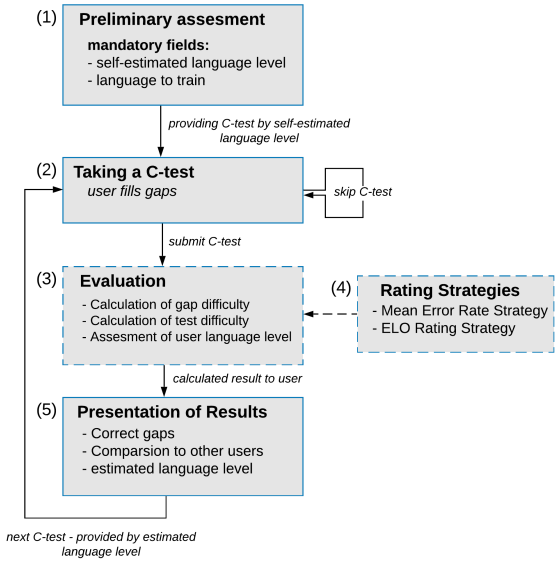

Figure 1: Interaction diagram of the user with the application. Non-interactive sections are marked with dashed lines.

user sees how many tests they have already passed, failed or skipped per level. The system uses this information to update the language proficiency of the user and provides them with new tests whose level matches that of the user.

Presentation of Results Once the user has submitted a c-test for evaluation, the results are presented. For wrongly answered gaps, the correct solution is shown. It is also shown how a user scored in relation to other users who took the same c-test. In addition, they are provided with the current assessment of their language level provided by the application. At this point, the user can either proceed to another c-test (or adjust their settings if they feel the test is too hard or too easy).

3 Admin perspective

The administrator is responsible for adding new tests with pre-set initial difficulty levels and can access the test results of the users. Figure 3 shows the administration view.

Adding Tests A c-test can be added in two ways: First, we support the export format of the C-Test Builder (Zesch et al., 2018), an authoring tool offering technical support for creating a c-test from raw text including automatic detection of Named Entities, numeric expression and other words that should be excluded from the gap scheme.[1] The tool also provides an automatic difficulty assessment as described in Beinborn et al. (2014), the result of which can be used as the initial value for the C-Test Collector. Second, text can be entered manually where those parts of a word that should appear as gaps are manually indicated using a bracket notation, e.g. *redun*{*dancy*}.

For each newly imported or created test, the admin has to provide the language, a title and an initial difficulty (i.e. a CEFR level). As we have already discussed that it is hard for teachers to correctly assign the difficulty level, this is only used at the beginning and the difficulty is adjusted as soon as more users have taken the test.

Test Overview The administrator is provided with a complete overview of all c-tests currently available in the application, as well as a graphical representation of the language distribution of tests (see Figure 3).

Test Details In the detailed view, information about a particular test is displayed. For each gap, an admin can inspect the distribution of errors made by test takers.

4 Scoring Tests and Users

Adjusting proficiency levels of users and difficulty levels of tests is a cold start problem: In the beginning, each user only has a self-assigned language proficiency and each test has only a rough difficulty level assigned by the administrator. This section describes how tests receive adjusted difficulty ratings and how users' proficiency assignments are adapted throughout their interaction with the system in order to present them with tests of a suitable difficulty.

Whenever a user completes a c-test, their current language level is adjusted. Similarly, the difficulty prediction of the c-test is refined with every new user taking the test, providing a larger basis for the difficulty estimation. We currently use two rating strategies (i) Mean Error Rate and, (ii) the Elo rating system (Elo, 1978).

4.1 Mean Error Rate

Under this strategy, the difficulty of a c-test is determined by the average error rates of the respective

[1] https://github.com/zesch/ctest-builder

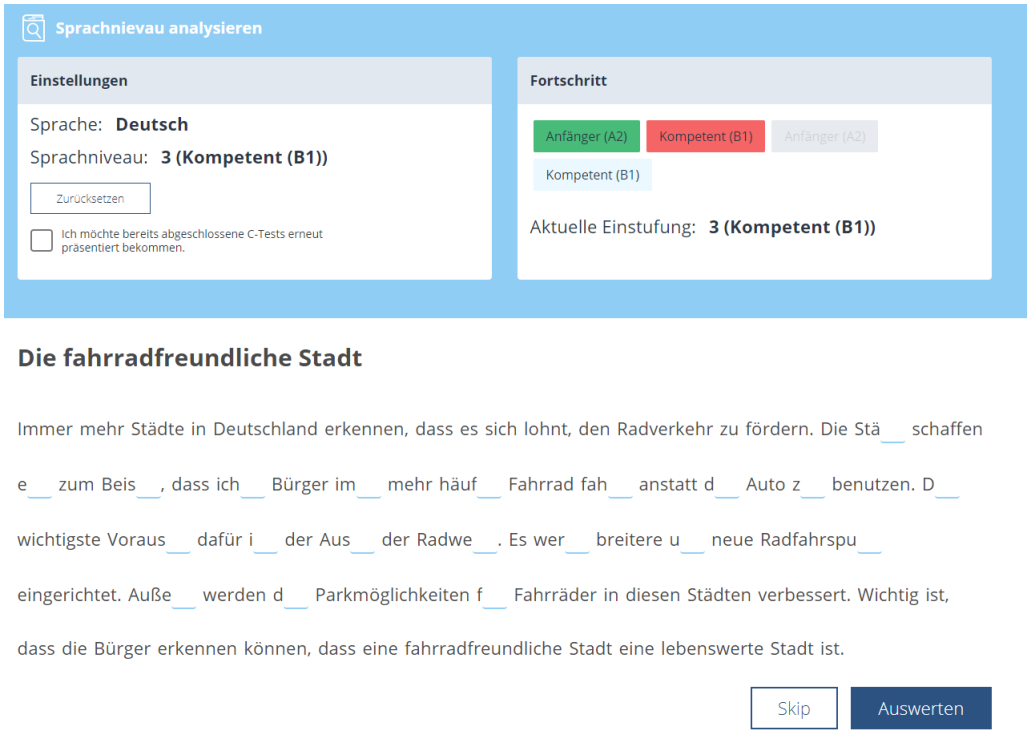

Figure 2: Screenshot from the application during the performance of a c-test. The three integration areas are visible: settings, progress and the actual test.

gaps.

$$MER = \frac{1}{n}\sum_{i=1}^{n} e_i$$

n is the number of gaps in the respective c-test and e is the difficulty of an individual gap. The difficulty of a gap is approximated via the percentage of users who can't solve that gap.

Under this strategy, the user proficiency is only indirectly adapted. A user is only presented with tests matching their current proficiency level. If the user correctly filled more than a certain threshold of gaps from the most recent n tests, their proficiency level is adapted to the next higher level. If they falls below a certain threshold, their proficiency level is adapted to the next lower level.

4.2 Elo Rating

The Elo strategy describes a system for calculating the level of a group of users relative to each other. Usually, the Elo method is applied in competitive sports such as chess or football (Hvattum and Arntzen, 2010). However, the method is also used in research (Lehmann and Wohlrabe, 2017) and adaptive educational settings similar to our case (Pelánek, 2016). In order to use the Elo strategy for our scenario, some adjustments have to be made that we describe in the following. We treat the user and the c-tests like opponents, so that each user and c-test receives a score. Informally speaking: if a user 'beats' a hard test (operationalized in our case as filling at least half of the gaps correctly), their Elo ranking will improve more than if a user correctly solves an easy test. Both ratings are in the range of $[1, 6] \subset \mathbb{N}$

More formally, we define the expectation of success E_U for the user as follows:

$$E_U = \frac{1}{1 + 10^{(R_T - R_U)}}$$

where R_U describes the rating of the user, R_T the rating of the c-test respectively and the success expectation for the test E_T is computed as $1 - E_U$.

Starting from the self-assigned user proficiency and admin assigned test difficulty, the ratings are updated as follows during the evaluation of a user's performance on a c-test:

$$R'_A = R_A + (S_A - E_A), for A \in \{T, U\}$$

R_A defines the old rating and R'_A the updated value. In order to translate the ratings back to the language levels, we round R'_A. In our version, S_A is 1 for winning and 0 for losing. The user "wins" if he has

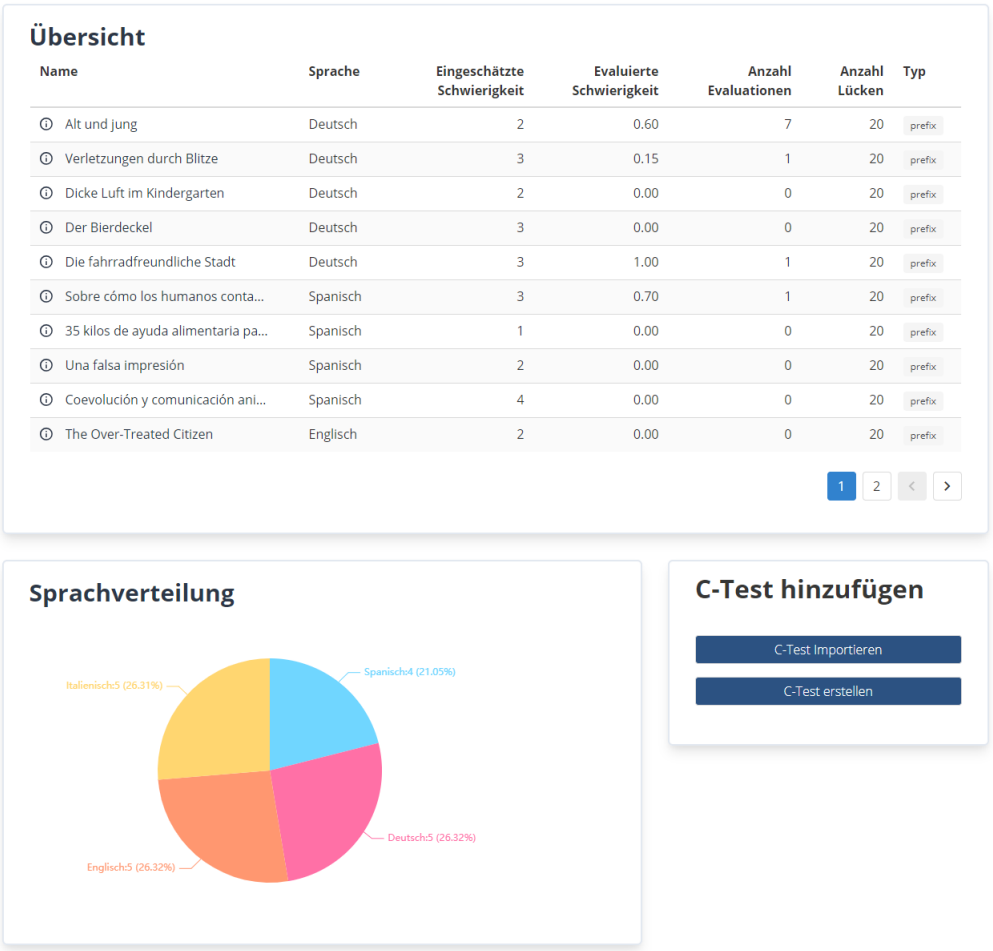

Figure 3: Screenshot of the administration panel. In addition to an overview of all c-tests (*Übersicht*), it is also possible to switch to a detailed view with e.g. language distributions (*Sprachverteilung*) and add new c-tests (*C-Test hinzufügen*.

completed at least 50% correctly. This means, the harder the test, the larger the update if the user can beat it.

5 Implementation Details

Our tool is implemented through a REST backend server and a separated frontend application. The source code is publicly available.[2]

The frontend of the application is provided through a VueJS application that communicates with the provided REST server. The interface provides the functionality to interact with the application and is more or less independent from the backend.

The provided REST server connects and mediates corresponding requests for the creation and evaluation of c-tests. The server was implemented using the Spring Boot Framework.

[2]https://gitlab.com/ctest-evaluator

6 Conclusion and outlook

We have presented an open-source implementation of a data collection tool that provides c-tests for language learners and collects training data for difficulty prediction tasks. The collected data, especially the individual incorrect solution attempts can also be used to study misspellings and cognates for learners of different L1s and L2s. So far, our prototype has only been tested with a small number of users. Next steps thus involve the deployment with a larger user base to start the actual data collection process for c-tests. For many languages, this will provide the first publicly available training data.

References

Lisa Beinborn, Torsten Zesch, and Iryna Gurevych. 2014. Predicting the difficulty of language proficiency tests. *Transactions of the Associ-*

ation for Computational Linguistics 2:517–530. https://www.aclweb.org/anthology/Q14-1040.

Council of Europe, editor. 2009. *Common European Framework of Reference for Languages: learning, teaching, assessment*. Cambridge Univ. Press [u.a.], Cambridge, 10. printing edition. OCLC: 837109490.

Arpad E. Elo. 1978. *The rating of chessplayers, past and present*. Arco Pub., New York. http://www.amazon.com/Rating-Chess-Players-Past-Present/dp/0668047216.

Ruediger Grotjahn. 2002. Konstruktion und Einsatz von C-Tests: Ein leitfaden für die Praxis pages 211–225.

Rüdiger Grotjahn. 2010. *Der C-Test: Beiträge aus der aktuellen Forschung - The C-Test: contributions from current research*. Peter Lang, Bern, Switzerland. https://www.peterlang.com/view/title/12445.

Lars Magnus Hvattum and Halvard Arntzen. 2010. Using ELO ratings for match result prediction in association football. *International Journal of Forecasting* 26(3):460–470. https://doi.org/10.1016/j.ijforecast.2009.10.002.

Ji-Ung Lee, Erik Schwan, and Christian M. Meyer. 2019. Manipulating the difficulty of C-tests. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, pages 360–370. https://doi.org/10.18653/v1/P19-1035.

Robert Lehmann and Klaus Wohlrabe. 2017. Who is the 'journal grand master'? a new ranking based on the elo rating system. *Journal of Informetrics* 11(3):800–809. https://doi.org/10.1016/j.joi.2017.05.004.

Radek Pelánek. 2016. Applications of the elo rating system in adaptive educational systems. *Computers & Education* 98:169–179.

Ulhich Raatz and Christine Klein-Braley. 1981. The c-test - a modification of the cloze procedure. In *Practice and Problems in Language Testing*. Colchester: University of Essex, Dept. of Language and Linguistics, pages 113–138.

Torsten Zesch, Andrea Horbach, Melanie Goggin, and Jennifer Wrede-Jackes. 2018. *A flexible online system for curating reduced redundancy language exercises and tests*, Research-publishing.net, pages 319–324. https://doi.org/10.14705/rpnet.2018.26.857.