DKPro Agreement: An Open-Source Java Library for Measuring Inter-Rater Agreement

Christian M. Meyer,[†] **Margot Mieskes,**[§†] **Christian Stab,**[†] **and Iryna Gurevych**[†‡]
[†] Ubiquitous Knowledge Processing Lab (UKP-TUDA)
Computer Science Department, Technische Universität Darmstadt
[‡] Ubiquitous Knowledge Processing Lab (UKP-DIPF)
German Institute for Educational Research
[§] Information Center for Education
German Institute for Educational Research (DIPF)
`http://www.ukp.tu-darmstadt.de`

Abstract

In this paper, we introduce a novel Java implementation of multiple inter-rater agreement measures, which we make available as open-source software. Besides assessing the reliability of coding tasks using S, π, κ, α, etc., we particularly support unitizing tasks by measuring α_{U} as the agreement of the boundaries of the identified annotation units. We provide a unified interface and data model for both tasks as well as multiple diagnostic devices for analyzing the results.

1 Introduction

Reliability is a necessary precondition for obtaining high-quality datasets and thus for drawing valid conclusions from an annotation study. Assessing the reliability by means of inter-rater agreement measures has been an established scientific practice in psychology (e.g., Cohen, 1960), medicine (Cicchetti et al., 1978), and content analysis (Krippendorff, 1980) for decades. In the computational linguistics and natural language processing community, reliability discussions have long been limited or completely ignored. It was not until Carletta's (1996) appeal that researchers started measuring the inter-rater agreement at a larger scale. However, there are still numerous papers published every year that lack a proper discussion of data quality. The reliability of the utilized datasets is, for example, not discussed at all in six out of the thirteen task description papers of SemEval-2013, and it remains shallow in another four papers, which do not provide a suitable inter-rater agreement figure.[1] A major reason for this is the limited availability of software components, which support the standard measures and are well-integrated with existing systems. In fact, many researchers currently rely on manual calculations, hasty implementations of single coefficients, or free online calculators that often lack documentation of the implementation details.

In this work, we present the novel Java-based software library *DKPro Agreement* for computing multiple inter-rater agreement measures using a shared interface and data model. For the first time, we provide a unified model for analyzing *coding* (i.e., assigning categories to fixed items) and *unitizing studies* (i.e., segmenting the data into codable units). By supporting these two fundamental annotation setups, our software can be used for analyzing many different annotation tasks including syntactic (e.g., part-of-speech tagging), semantic (e.g., word sense assignment, keyphrase identification), and discourse annotation tasks (e.g., dialogue act tagging). We particularly provide diagnostic devices for analyzing systematic disagreement, which is often overlooked in reliability discussions. DKPro Agreement is available as open-source software, thoroughly tested on a wide range of examples, and well-documented for getting started quickly.[2] Our implementation is targeted at scientists and software developers working in Java, including the large communities around the major Java-based toolkits *OpenNLP*, *DKPro*, and *GATE*.[3] The software integrates more easily with existing Java applications than statistics software such as *Octave*, *R*, or *SPSS*, and its usage requires a less pronounced statistics background.

[1]All SemEval task descriptions (`http://www.cs.york.ac.uk/semeval-2013`) have been discussed among the authors.

[2]DKPro Agreement is part of the DKPro Statistics library. The project is available from `https://code.google.com/p/dkpro-statistics/` and licensed under the Apache License 2.0.

[3]`http://opennlp.apache.org`, `http://code.google.com/p/dkpro-core-asl`, `http://gate.ac.uk`

2 Related Work

Reliability is the subject of an extensive body of literature. Artstein and Poesio (2008) and Krippendorff (1980) give a general introduction to this topic. Implementations of inter-rater agreement measures exist both as stand-alone and as online tools (e.g., http://uni-leipzig.de/~jenderek/tool/tool.htm, http://terpconnect.umd.edu/~dchoy/thesis/Kappa). Most of them focus on one type of agreement measure (e.g., http://vassarstats.net/kappa.html and Randolph (2005) for κ as well as http://ron.artstein.org/software.html for α). Others are limited to a few different measures (e.g., https://mlnl.net/jg/software/ira and http://dfreelon.org/utils/recalfront/recal3). Frameworks offering a wider range of measures are either commercial with a high price tag (e.g., http://www.medcalc.org/manual/kappa.php) or limited to specific platforms (e.g., http://www.agreestat.com/agreestat.html). The situation is even worse for unitizing studies, for which we are only aware of Perry and Krippendorff's (2013) implementation available at http://www.gabriela.trindade.nom.br/2013/02/calculating-alpha-d-and-alpha-u/. To the best of our knowledge, there is no software library providing a large variety of agreement measures and diagnostic devices, which covers both coding and unitizing studies and which integrates well with existing systems.

3 Implementation

The standard workflow for using DKPro Agreement is (1) representing the annotated data using our unified data model, (2) measuring the agreement among the individual raters, and (3) analyzing the results using diagnostic devices and visualizations.

Data model. We provide the Java interface `IAnnotationStudy` as the basic representation of the annotated data. An annotation study consists of a set of *raters*, a set of codable *units*, and a set of *categories*, which may be used by the raters to code a unit. The categories do not have a specific type, such that any Java object (including integers and enums) may be used without any extensions.

Following Krippendorff's (1980) terminology, we distinguish two basic annotation setups: In *coding studies*, the raters receive a set of *annotation items* with fixed boundaries (e.g., full articles from a newspaper), which each of them should code ("annotate") with one of the categories (e.g., politics, economics). We consider each rater's annotation of an item a single *annotation unit*. In *unitizing studies*, the raters are asked to identify the annotation units themselves by marking their boundaries (e.g., highlighting key phrases). Depending on the task definition, the identified units may be coded with one of multiple categories or just distinguish identified segments from so-called *gaps* between these segments.

Figure 1 illustrates this data model for both setups. The framed boxes show the annotation units and the categories assigned to them. In coding studies, the units with identical index (i.e., the columns) yield the annotation items. In unitizing studies, the units may be positioned arbitrarily within the continuum. We represent missing annotations as empty boxes (coding studies) and as horizontal lines between the identified units (unitizing studies).

Software developers can either instantiate our default implementation (e.g., by reading data from flat files or databases) or implement the provided Java interfaces in order to reuse their own data model. For coding studies, there is an `addItem` method with a *varargs* parameter (i.e., a method taking an arbitrary number of parameters) for specifying the annotation of each rater for a certain item. The line `study.addItem("B", "C", null, "B")` indicates that four raters coded an item with the categories B, C, *null*, and B. We use *null* to represent missing annotations. Similarly, unitizing studies provide an `addUnit` method, which takes the boundaries and the category assigned to the unit by a certain rater. The line `study.addUnit(10, 4, 2, "A")` indicates, for instance, a unit of length 4, which starts at position 10 and which has been annotated as category A by rater 2.

Coding measures. Table 1 shows an overview of the inter-rater agreement measures currently available in DKPro Agreement. Artstein and Poesio (2008) give an overview of these measures. While the *percentage agreement* simply divides the number of agreements by the item count, all other measures perform a *chance correction*. A major difference is the assumed probability distribution for the expected agreement, which is considered different for each study and rater (i.e., *rater-specific*), the same for all

Measure	Type	Raters	Chance correction	Weighted
Percentage agreement	coding	≥ 2	–	–
Bennett et al.'s S (1954)	coding	2	uniform	–
Scott's π (1955)	coding	2	study-specific	–
Cohen's κ (1960)	coding	2	rater-specific	–
Randolph's κ (2005) [multi-S]	coding	≥ 2	uniform	–
Fleiss's κ (1971) [multi-π]	coding	≥ 2	study-specific	–
Hubert's κ (1977) [multi-κ]	coding	≥ 2	rater-specific	–
Krippendorff's α (1980)	coding	≥ 2	study-specific	✓
Cohen's weighted κ_{w} (1968)	coding	≥ 2	rater-specific	✓
Krippendorff's α_{u} (1995)	unitizing	≥ 2	study-specific	–

Table 1: Implemented inter-rater agreement measures

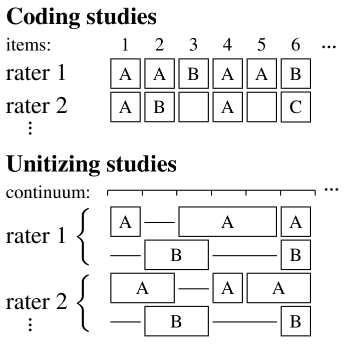

Figure 1: Data model

raters (*study-specific*), or the same for all studies and raters (*uniform*). As all of these measures are used in the literature, we need implementations for each of them to be able to compare different results. This is particularly important for κ measures, because many different definitions exist. Cohen's κ (1960) is, for instance, often compared to Fleiss's κ (1971), although both measures assume a different probability distribution. Since all measures implement a standardized interface, different results can be easily compared using our software. The line `new PercentageAgreement(study).calculateAgreement()` returns, for instance, the percentage agreement of the given study, while the line `new FleissKappaAgreement(study).calculateAgreement()` returns Fleiss's κ (1971) for the very same study.

Most early measures consider an agreement if, and only if, the categories assigned to an item are identical. *Weighted measures* allow for defining a *distance function* that expresses the similarity of two categories. We provide distance functions for nominal, ordinal, interval, and ratio scales (Krippendorff, 1980), as well as the MASI distance function for set-valued data (Passonneau, 2006). Set annotations are an example for a more complex type of category, as they facilitate assigning multiple categories to a given unit. Additionally, researchers can easily define new study-specific distance functions.

Unitizing measures. Although unitizing has long been identified as a major issue of reliability analysis (cf. Auld and Whitea, 1956), there are so far only few formalizations. The most elaborate measure is Krippendorff's α_U (1995), which is based on a distance function for comparing units with identical, overlapping, or disjoint boundaries. As a model for expected disagreement, α_U considers all possible unitizations for the given continuum and raters. Thereby, α_U becomes fully compatible with Krippendorff's α (1980) for coding tasks. Due to the lack of implementations and the complex statistics, α_U is almost never reported in the literature. While our implementation follows the original definition, it does not require to explicitly encode the gaps, because we generate them automatically. This simplifies the usage of this measure substantially, since, for example, UIMA[4] annotations can be directly used to represent the annotation units. While the original definition focuses on only one category, we additionally support Krippendorff's later definition of an aggregating α_U over all categories (Krippendorff, 2004).

Analysis. The raw inter-rater agreement scores are useful for comparing multiple annotation studies with each other, but they are of limited help for diagnosing disagreement and potential systematic issues with certain categories, items, or raters. This is why we provide interfaces for measuring the agreement of a specific category, item, or rater. Fleiss (1971) defines, for instance, a category-specific κ_c that we realize in our software. The same holds for the unitizing measure α_U, which also provides a category-specific agreement score. Besides measuring a rater-specific agreement score, DKPro Agreement facilitates the computation of pairwise inter-rater agreement in order to identify the pair of raters with the highest and lowest agreement. Finally, each measure can return some of its intermediate results, for example, the expected agreement P_e of Scott's π (1955).

Another means of analysis is to display the annotation units and the disagreement among the raters. Coding studies can be displayed as a *coincidence table* or as a *reliability matrix* (Krippendorff, 1980). For studies with two raters, our software also allows printing a *contingency table*. In addition to that,

[4]Unstructured Information Management Architecture, http://uima.apache.org

we provide a formatter for the *weighing matrix* of a distance function and a basic visualization of the continuum of annotation units in unitizing studies – similar to the representation in Figure 1.

Code example. Consider the coding study described by Krippendorff (1980, p. 139) consisting of nine annotation items that have been categorized as category 1, 2, 3, or 4 by three raters. We can (re-)analyze this study with DKPro Agreement using the following Java code:

```
CodingAnnotationStudy study = new CodingAnnotationStudy(3); ❶

study.addItem(1, 1, 1); study.addItem(1, 2, 2); study.addItem(2, 2, 2); ❷
study.addItem(4, 4, 4); study.addItem(1, 4, 4); study.addItem(2, 2, 2);
study.addItem(1, 2, 3); study.addItem(3, 3, 3); study.addItem(2, 2, 2);

PercentageAgreement pa = new PercentageAgreement(study); ❸
System.out.println(pa.calculateAgreement());

KrippendorffAlphaAgreement alpha = new KrippendorffAlphaAgreement(
    study, new NominalDistanceFunction());
System.out.println(alpha.calculateObservedDisagreement());
System.out.println(alpha.calculateExpectedDisagreement());
System.out.println(alpha.calculateAgreement());

System.out.println(alpha.calculateCategoryAgreement(1)); ❹
System.out.println(alpha.calculateCategoryAgreement(2));

new CoincidenceMatrixPrinter().print(System.out, study); ❺
```

The first step is ❶ the instanciation of an `IAnnotationStudy` for the three human raters. Then, ❷ we add all $3 \cdot 9 = 27$ annotation units to the study by providing the category chosen by each rater to code the nine annotation items. Note that most reliability analyses would read the raters's decisions from a file, database, or other data structure rather than typing them in manually like in this example. Once the data model is complete, ❸ we calculate the inter-rater agreement. Following the original publication, we calculate the raw agreement and Krippendorff's α (1980) and thus print 0.740 (percentage agreement), 0.259 (observed disagreement D_{O}), 0.724 (expected disagreement D_{E}), and 0.642 (α coefficient). Finally, ❹ we analyze the agreement by calculating the category-specific α for the categories 1 and 2 yielding a system output of 0.381 and 0.711, and ❺ we print a coincidence matrix on the system console.

4 Evaluation and Publication

Automatic tests. Even though the agreement measures are clearly defined in the literature, their implementation is error-prone due to the varying notations and the required changes to take efficiency and numerical stability into account. A single confused index (e.g., p_{ij} instead of p_{ji}) could easily yield invalid conclusions for many studies. This is why we provide 61 unit tests to evaluate the correctness of our implementation. Besides manually devised examples, we use 46 examples from the literature (e.g., from Krippendorff, 1980). All examples contain references to their original documentation.

Numerical stability. We especially test the analysis of larger annotation studies, which raise issues of numerical stability. When scaling the example by Artstein and Poesio (2008, p. 558) with factor 500, an implementation potentially returns a Cohen's κ of 0.82, although it is only 0.35. This phenomenon is due to arithmetic overflows, instable division operations, and rounding errors. Krippendorff's α_{U} depends, for instance, on the cubic factor $2l_{hj}^3$, which can raise such issues even for rather small studies. Where necessary, we use logarithms or Java's `BigDecimal` type to ensure accurate results.

Open-source software. We publish our implementation as open-source software under the Apache License. By providing access to our source code, researchers can learn how the measures work and how the software is used. Moreover, they can easily contribute in order to extend the library and evaluate its correctness in a peer review, which is an essential step to establish the credibility of the software. Finally, our choice of a free license should ease (re-)using the software in many projects and thus facilitate the reproduction and comparison of annotation results, which often falls short in our community.

Documentation. DKPro Agreement is fully documented using Javadoc comments. In addition to that, we provide a general introduction and a getting-started tutorial on the project page, which also points the users to our numerous usage examples in the form of test cases.

5 Conclusion and Future Work

We have presented an open-source Java software for measuring inter-rater agreement of coding and unitizing studies. In future work, we plan to provide additional diagnostic devices and elaborated visualizations (such as Hinton diagrams) and to integrate our measures with existing annotation workbenches. An early software version has, for example, already been used in the *CSniper* (Eckart de Castilho et al., 2012) and *WebAnno* (Yimam et al., 2013) systems. We plan to extent this to other systems and also make use of the newly introduced unitizing setup.

Acknowledgments

This work has been supported by the Volkswagen Foundation as part of the Lichtenberg-Professorship Program under grant No. I/82806.

References

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

Frank Auld, Jr and Alice M. Whitea. 1956. Rules for Dividing Interviews Into Sentences. *The Journal of Psychology: Interdisciplinary and Applied*, 42(2):273–281.

Edward M. Bennett, R. Alpert, and A. C. Goldstein. 1954. Communications Through Limited Response Questioning. *Public Opinion Quarterly*, 18(3):303–308.

Jean Carletta. 1996. Assessing Agreement on Classification Tasks: The Kappa Statistic. *Computational Linguistics*, 22(2):249–254.

Domenic V. Cicchetti, Chinyu Lee, Alan F. Fontana, and Barbara Noel Dowds. 1978. A Computer Program for Assessing Specific Category Rater Agreement for Qualitative Data. *Educational and Psychological Measurement*, 38(3):805–813.

Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37–46.

Jacob Cohen. 1968. Weighted kappa: Nominal scale agreement with provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70(4):213–220.

Richard Eckart de Castilho, Sabine Bartsch, and Iryna Gurevych. 2012. CSniper – Annotation-by-query for non-canonical constructions in large corpora. In *Proceedings of the 50th Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 85–90, Jeju Island, Korea.

Joseph L. Fleiss. 1971. Measuring Nominal Scale Agreement among many Raters. *Psychological Bulletin*, 76(5):378–382.

Lawrence Hubert. 1977. Kappa revisited. *Psychological Bulletin*, 84(2):289–297.

Klaus Krippendorff. 1980. *Content Analysis: An Introduction to Its Methodology*. Beverly Hills, CA: Sage Publications.

Klaus Krippendorff. 1995. On the reliability of unitizing contiguous data. *Sociological Methodology*, 25:47–76. Published for American Sociological Association.

Klaus Krippendorff. 2004. *Content Analysis: An Introduction to Its Methodology*. Thousand Oaks, CA: Sage Publications, 2nd edition.

Rebecca J. Passonneau. 2006. Measuring agreement on set-valued items (MASI) for semantic and pragmatic annotation. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, pages 831–836, Genoa, Italy.

Gabriela Trindade Perry and Klaus Krippendorff. 2013. On the reliability of identifying design moves in protocol analysis. *Design Studies*, 34(5):612–635.

Justus J. Randolph. 2005. Free-marginal multirater kappa (multirater κ_{free}): An alternative to Fleiss' fixed-marginal multirater kappa. In *Proceedings of the 5th Joensuu University Learning and Instruction Symposium*, Joensuu, Finland.

William A. Scott. 1955. Reliability of Content Analysis: The Case of Nominal Scale Coding. *Public Opinion Quaterly*, 19(3):321–325.

Seid Muhie Yimam, Iryna Gurevych, Richard Eckart de Castilho, and Chris Biemann. 2013. WebAnno: A Flexible, Web-based and Visually Supported System for Distributed Annotations. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 1–6, Sofia, Bulgaria.