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 (UKP) Lab, Technische Universität Darmstadt / German Institute for Educational Research

Summary

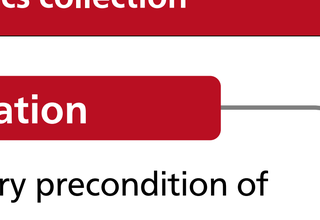
DKPro Agreement is an open-licensed Java library for computing inter-rater agreement using a shared interface and data model.

Highlights:

- **Support for all commonly used inter-rater agreement measures**
- **Calculation of multiple coefficients using the same data model**
- **Both coding and unitizing setups are possible**
- **Multiple diagnostic devices and visual aids for analyzing disagreement**
- **Thoroughly tested on a wide range of examples from the literature**
  - **over 60 test cases for annotation studies including citation of original source**
- **Available as open source software under the Apache License 2.0 (ASL)**
  - **Extensions and comments welcome!**
- **Integrates well with existing Java-based NLP frameworks**
- **Ready-to-use via Maven Central – simply specify the dependency:**
  - **groupId: de.tudarmstadt.ukp.dkpro.statistics**
  - **artifactId: dkpro-statistics-agreement**
  - **version: 2.0.0**
- **Part of DKPro Statistics collection**

Motivation

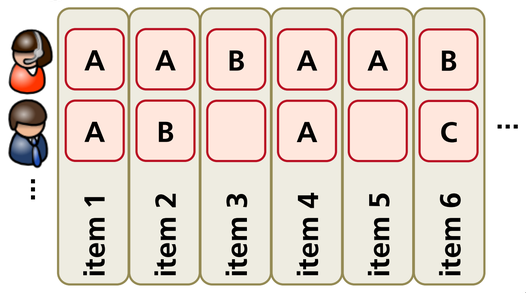
- Reliability is a necessary precondition of high quality datasets
- Long tradition of assessing inter-rater agreement in psychology, medicine, content analysis
- In NLP/CL often ignored or limited
- Researchers rely on
  - manual calculations
  - hasty implementation
  - insufficiently documented online calculators
- Measures are often not comparable
- Urgent need for software that
  - implements the most important measures
  - allows for diagnosing disagreement
  - integrates with existing projects and annotation workbenches (e.g., WebAnno, CSniper)



Coding Setup

Raters assign categories to fixed items.

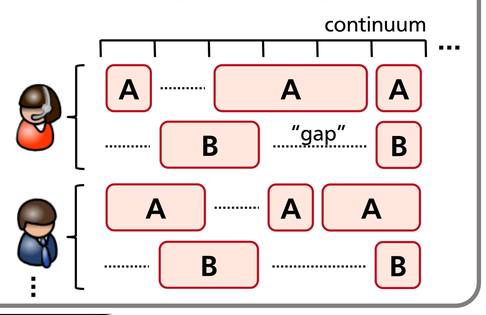
- Document classification
- POS tagging
- Dialog act tagging
- etc.



Unitizing Setup

Raters segment data into codable units.

- Keyphrase identification
- Argument tagging
- Disfluencies
- etc.



DKPro Agreement
http://code.google.com/p/dkpro-statistics/

Step 1: Represent the Annotated Data

Define annotations manually

```
study.addItem(Object… <annotations>)
```

Code Example:

```
study.addItem("A", "A", "B", "A");
study.addItem("B", "B", "B", "B");
study.addItem("B", "C", null, "B");

study.addUnit(<offset>, <length>,
     <rater>, <category>)
```

Code Example:

```
study.addUnit(10, 4, 2, "A");
study.addUnit(20, 1, 1, "B");
study.addUnit(20, 3, 2, "B");
```

or load from flat files/DB

Code Example:

```
CodingAnnotationStudy study = new
     CodingAnnotationStudy(3);
BufferedReader reader =
     new BufferedReader(
     new FileReader("flatfile.tsv"));
String line;
while ((line = reader.readLine())
     != null) {
  study.addItemAsArray(
     line.split("\t"));
}
reader.close();
```

or use UIMA annotations

Code Example:

```
UnitizingAnnotationStudy study =
     new UnitizingAnnotationStudy(2,
     jcas.getDocumentText().length());
for (Annotation a : JCasUtil.select(
     jcas, Annotation.class)) {
  study.addUnit(a.getBegin(),
       a.getEnd() - a.getBegin(),
       a.getRaterIdx(), true);
}
```

or reuse your own data model by implementing available interfaces.

Step 2: Measure the Inter-Rater Agreement

Available coefficients:

Measure	Type	Raters	Chance-corr.	Weighted
Percentage agreement p	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	–

Code example:

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

FleissKappaAgreement kappa =
     new FleissKappaAgreement(study);
System.out.println(kappa.calculateAgreement());

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

Step 3: Analyze the Disagreement

Raw agreement scores are of limited help for diagnosing the main sources of disagreement. DKPro Agreement provides multiple diagnostic devices:

Agreement insights

- Observed agreement
- Expected agreement
- Rater-specific agreement
- Category-specific agreement
- Item-specific agreement

		items						
		1	**2**	**3**	**4**	**5**	**6**	**∑**
raters		A	A	B	A	A	B	
		A	B		A		C	
categories	A	2	1		2	1		**6**
	B		1	1			1	**3**
	C						1	**1**

$p = 0.50$ | $\kappa = 0.08$ | $\alpha = 0.18$
$\alpha(A) = 0.39$ | $\alpha(B) = -0.22$ | $\alpha(C) = 0.00$

Formatted output and visual aids

- Coincidence matrix
- Contingency matrix
- Reliability matrix
- Continuum of a unitizing study
- Planned: Hinton diagrams

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

http://www.ukp.tu-darmstadt.de