`pyannote.metrics`: a toolkit for reproducible evaluation, diagnostic, and error analysis of speaker diarization systems

Hervé Bredin

CNRS, LIMSI, Université Paris-Saclay, Orsay, France

bredin+pyannote@limsi.fr

Abstract

`pyannote.metrics` is an open-source Python library aimed at researchers working in the wide area of speaker diarization. It provides a command line interface (CLI) to improve reproducibility and comparison of speaker diarization research results. Through its application programming interface (API), a large set of evaluation metrics is available for diagnostic purposes of all modules of typical speaker diarization pipelines (speech activity detection, speaker change detection, clustering, and identification). Finally, thanks to visualization capabilities, we show that it can also be used for detailed error analysis purposes. `pyannote.metrics` can be downloaded from `http://pyannote.github.io`.

Index Terms: evaluation, speaker diarization, reproducible research, open-source software

1. Introduction

Speaker diarization is the task of partitioning an audio stream into homogeneous temporal segments according to the identity of the speaker. Automatic speech transcription also benefits from speaker diarization to address the question "who speaks what?". Resulting augmented (or "rich") transcription can be very useful for multimedia documents structuring and indexing.

Thanks to open initiatives such as the series of NIST "Rich Transcription" evaluations [1], or ESTER [2] and ETAPE [3] benchmarks, the state-of-the-art for speaker diarization has achieved significant improvement since 2000. Despite addressing the same task, these initiatives used different evaluation metrics, different implementations of these metrics [4, 5], all provided as standalone (Perl or Lua) command line tools.

In this paper, we introduce `pyannote.metrics`, an open-source Python library for reproducible evaluation, diagnostic, and error analysis of speaker diarization systems. Python is currently being adopted by a growing number of researchers in speaker identification [6, 7], machine learning [8], or deep learning [9]. Evaluating speaker diarization using the same language as the one used for developing, training, and testing a system has several advantages. In particular, it may be very convenient when the actual evaluation metric serves as a cost function to minimize during training.

For the sake of completeness, `pyannote.metrics` can also be used as a command line tool (described in Section 2) to compute the *de facto* standard *diarization error rate* NIST implementation. Inspired by Bob [7] database package paradigm, `pyannote.metrics` rely on standardized database interfaces to ensure reproducibility and fair comparison of speaker diarization systems. It also provides a large collections of additional evaluation metrics (summarized in Figure 1) that can be used for diagnostic purposes, either using the command line tool, or directly as a Python module. Those additional metrics are described, discussed, and compared in Section 3. Finally, another strength of `pyannote.metrics` lies in its advanced visualization and error analysis features. This is introduced in Section 4.

2. Reproducible evaluation

There are two main issues that may arise with results reported in the literature. Firstly, even though the same public datasets are used, the actual evaluation protocol may differ slightly from one paper to another. Secondly, the implementation of the reported evaluation metrics may also differ. The first objective of the `pyannote.metrics` library is to address these two problems, and provide a convenient way for researchers to evaluate their approaches in a reproducible and comparable manner. Figure 2 shows an example use of the command line interface that is provided to solve this problem.

2.1. Tasks

Not only can `pyannote.metrics.py` command line tool be used to compute the diarization error rate using NIST implementation, one can also evaluate the typical four submodules used in most speaker diarization systems [10], depicted in Figure 1. Practically, the first positional argument (*e.g.* `diarization` in Figure 2) is a flag indicating which task should be evaluated. Apart from the `diarization` flag that is used for evaluating speaker diarization results, other available flags are `detection` (speech activity detection), `segmentation` (speaker change detection), and `identification` (supervised speaker identification). Depending on the task, a different set of evaluation metrics is computed, which are listed in Section 3.

2.2. Datasets and protocols

`pyannote.metrics` provides an easy way to ensure the same protocol (*i.e.* manual groundtruth and training/development/test split) is used for evaluation. Internally, it relies on a collection of Python packages that all derive from the `pyannote.database` main package, that provides a convenient API to define training/development/test splits, along with groundtruth annotations. As of March 2017, `pyannote.database` packages exist for the ETAPE corpus [3], the REPERE corpus [11, 12], and the AMI corpus [13]. As more people contribute new `pyannote.database` packages, they will be added to the `pyannote` ecosystem. In Figure 2, the development set of the *TV* evaluation protocol of the ETAPE dataset is used. Results are both reported for each file in the selected subset, and aggregated into one final metric value (*cf.* line starting with `TOTAL`).

2.3. File formats

While the `MDTM` [4] file format is used in this example, several other file formats are available (and can be contributed) thanks

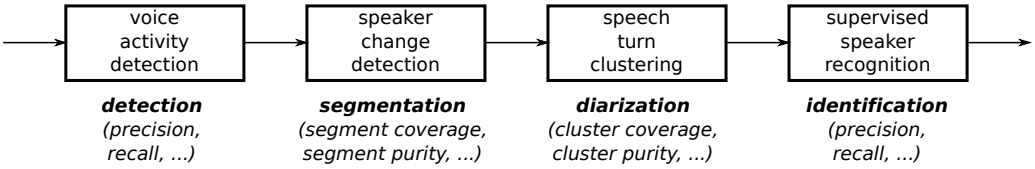

Figure 1: *A typical pipeline for speaker diarization, aligned with the list of available evaluation metrics*

```
$ pyannote.metrics.py diarization --subset=development Etape.SpeakerDiarization.TV hypothesis.mdtm
```

Diarization (collar = 0 ms)	error	purity	coverage	total	correct	%	fa.	%	miss.	%	conf.	%
BFMTV_BFMStory_2011-03-17_175900	14.64	94.74	90.00	2582.08	2300.22	89.08	96.16	3.72	80.14	3.10	201.72	7.81
LCP_CaVousRegarde_2011-02-17_204700	17.80	89.13	86.90	3280.72	2848.42	86.82	151.78	4.63	208.29	6.35	224.01	6.83
LCP_EntreLesLignes_2011-03-18_192900	23.46	79.52	79.03	1704.97	1337.80	78.46	32.89	1.93	157.14	9.22	210.03	12.32
LCP_EntreLesLignes_2011-03-25_192900	26.75	76.97	75.86	1704.13	1292.83	75.86	44.61	2.62	158.38	9.29	252.92	14.84
LCP_PileEtFace_2011-03-17_192900	10.73	93.33	92.30	1611.49	1487.32	92.30	48.73	3.02	55.49	3.44	68.67	4.26
LCP_TopQuestions_2011-03-23_213900	18.28	98.25	94.20	727.26	668.65	91.94	74.36	10.22	16.41	2.26	42.20	5.80
LCP_TopQuestions_2011-04-05_213900	27.97	97.95	79.81	818.03	638.68	78.08	49.45	6.04	17.46	2.13	161.89	19.79
TV8_LaPlaceDuVillage_2011-03-14_172834	21.43	92.89	89.64	996.12	892.04	89.55	109.36	10.98	11.80	1.18	92.28	9.26
TV8_LaPlaceDuVillage_2011-03-21_201334	66.23	77.24	70.64	1296.86	691.76	53.34	253.80	19.57	29.16	2.25	575.95	44.41
TOTAL	23.27	88.18	84.55	14721.65	12157.71	82.58	861.14	5.85	734.28	4.99	1829.67	12.43

Figure 2: *`pyannote-metrics.py` example output*

to the internal use of the `pyannote.parser` package.

3. Diagnostic

A typical speaker diarization pipeline is depicted in Figure 1. The first step is usually dedicated to speech activity detection, where the objective is to remove all non-speech regions. Then, speaker change detection aims at segmenting speech regions into homogeneous segments. The subsequent clustering step tries to group those speech segments according to the identity of the speaker. Finally, an optional supervised classification step may be applied to actually identify every speaker cluster in a supervised way.

Looking at the final performance of the system is usually not enough for diagnostic purposes. In particular, it is often necessary to evaluate the performance of each module separately to identify their strenght and weakness, or to estimate the influence of their errors on the complete pipeline. This section provides the list of metrics that were implemented in `pyannote.metrics` with that very goal in mind.

3.1. Detection

Speech activity detection modules can be evaluated using, detection error rate, precision, and recall.

$$\text{detection error rate} = \frac{\text{false alarm} + \text{missed detection}}{\text{total}}$$

where false alarm is the duration of non-speech incorrectly classified as speech, missed detection is the duration of speech incorrectly classified as non-speech, and total is the total duration of speech in the reference. Note that these metrics do not take overlapping speech into account. In other words, overlapping speech regions are counted only once.

3.2. Segmentation

As depicted in Figure 3, (speaker) change detection modules can be evaluated using two pairs of dual metrics: precision and recall, or purity and coverage. Precision and recall are standard metrics based on the number of correctly detected speaker boundaries. In Figure 3, recall is 75% because 3 out of 4 reference boundaries were correctly detected, and precision is 100%

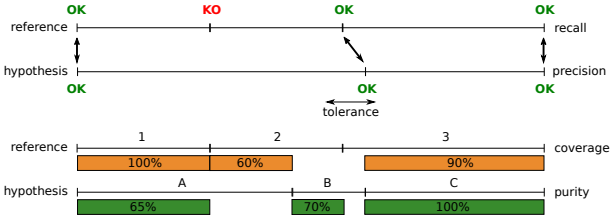

Figure 3: *Segmentation metrics available in `pyannote.metrics`*

because all hypothesized boundaries are correct. The main weakness of that pair of metrics (and their combination into a f-score) is that it is very sensitive to the *tolerance* parameter, *i.e.* the maximum distance between two boundaries for them to be matched. From one segmentation paper to another, authors may used very different values, thus making the approaches difficult to compare.

Instead, we think that segment-wise purity and coverage should be used instead. They have several advantages over precision and recall, including the fact that they do not depend on any *tolerance* parameter, and that they directly relate to the cluster-wise purity and coverage used for evaluating speaker diarization. Segment-wise coverage is computed for each segment in the reference as the ratio of the duration of the intersection with the most co-occurring hypothesis segment and the duration of the reference segment. For instance, coverage for reference segment 1 is 100% because it is entirely covered by hypothesis segment A. Purity is the dual metric that indicates how *pure* hypothesis segments are. For instance, segment A is only 65% pure because it is covered at 65% by segment 1 and 35% by segment 2. The final values are duration-weighted average over each segment.

3.3. Diarization

Diarization error rate (DER) is the *de facto* standard metric for evaluating and comparing speaker diarization systems. It is defined as follows:

$$\text{DER} = \frac{\text{false alarm} + \text{missed detection} + \text{confusion}}{\text{total}}$$

where false alarm is the duration of non-speech incorrectly classified as speech, missed detection is the duration of speech incorrectly classified as non-speech, confusion is the duration of speaker confusion, and total is the total duration of speech in the reference. Note that this metric does take overlapping speech into account, potentially leading to increased missed detection in case the speaker diarization system does not include an overlapping speech detection module.

3.3.1. *"Optimal"* vs. *"greedy"*

Two implementations of the diarization error rate are available (optimal and greedy), depending on how the one-to-one mapping between reference and hypothesized speakers is computed. The *optimal* version uses the Hungarian algorithm [14] to compute the mapping that minimize the confusion term, while the *greedy* version operates in a greedy manner, mapping reference and hypothesized speakers iteratively, by decreasing value of their cooccurrence duration. In practice, the *greedy* version is much faster than the *optimal* one, especially for files with a large number of speakers – though it may slightly over-estimate the value of the diarization error rate.

3.3.2. *Purity and coverage*

While the diarization error rate provides a convenient way to compare different diarization approaches, it is usually not enough to understand the type of errors commited by the system. Purity [15] and coverage [16] are two dual evaluation metrics that provide additional insight on the behavior of the system. They are defined as follows:

$$\text{purity} = \frac{\sum_{\text{cluster}} \max_{\text{speaker}} |\text{cluster} \cap \text{speaker}|}{\sum_{\text{cluster}} |\text{cluster}|}$$

$$\text{coverage} = \frac{\sum_{\text{speaker}} \max_{\text{cluster}} |\text{speaker} \cap \text{cluster}|}{\sum_{\text{speaker}} |\text{speaker}|}$$

where $|\text{speaker}|$ (respectively $|\text{cluster}|$) is the speech duration of this particular reference speaker (resp. hypothesized cluster), and $|\text{speaker} \cap \text{cluster}|$ is the duration of their intersection. Over-segmented results (*e.g.* too many speaker clusters) tend to lead to high purity and low coverage, while under-segmented results (*e.g.* when two speakers are merged into one large cluster) lead to low purity and higher coverage.

3.3.3. *Use case*

Figure 4 depicts the evolution of LIMSI multi-stage speaker diarization system [17] applied on the ETAPE dataset [3]. It is roughly made of four consecutive modules (segmentation, BIC clustering, Viterbi resegmentation, and CLR clustering). From the upper part of the figure (DER as a function of the module), it is clear that each module improves the output of the previous one. Yet, the lower part of the figure clarifies the role of each module. BIC clustering tends to increase the size of the speaker clusters, at the expense of purity (-7%). Viterbi resegmentation addresses this limitation and greatly improves cluster purity ($+5\%$), with very little impact on the actual cluster coverage ($+2\%$). Finally, CLR clustering brings an additional $+5\%$ coverage improvement.

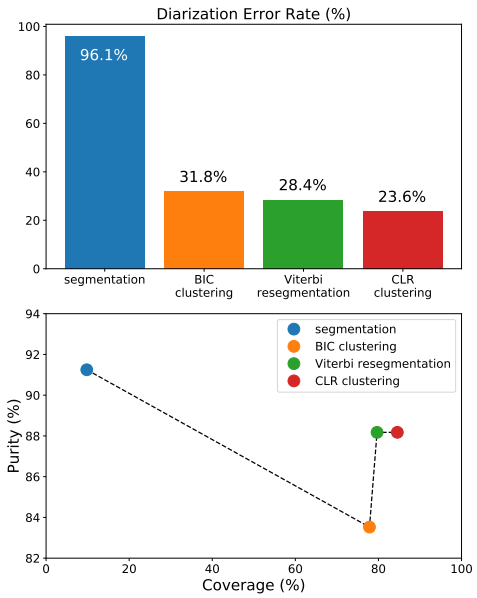

Figure 4: *Diagnosing LIMSI multi-stage speaker diarization system [17]*

3.4. Identification

In case prior speaker models are available, the speech turn clustering module in Figure 1 may be followed by a supervised speaker recognition module for cluster-wise supervised classification. `pyannote.metrics` also provides a collection of evaluation metrics for this identification task. This includes precision, recall, and identification error rate (IER):

$$\text{IER} = \frac{\text{false alarm} + \text{missed detection} + \text{confusion}}{\text{total}}$$

which is similar to the diarization error rate (DER) introduced previously, except that the `confusion` term is computed directly by comparing reference and hypothesis labels, and does not rely on a prior one-to-one matching.

3.5. Collar and evaluation map

Because manual annotations cannot be precise at the audio sample level, it is common in speaker diarization research to remove from evaluation a 500ms collar around each speaker turn boundary (250ms before and after). Most of the metrics available in `pyannote.metrics` support a *collar* parameter, which defaults to 0.

Moreover, though audio files can always be processed entirely (from beginning to end), there are cases where reference annotations are only available for some regions of the audio files. All metrics support the provision of an *evaluation map* [4] that indicate which part of the audio file should be evaluated.

4. Visual error analysis

As useful as those metrics can be, it is often necessary to have a closer look at the actual output of each module, for error analysis or visual inspection. Existing audio annotation tools

such as transcriber [18] or Praat [19] may be used for that purpose. However, they require the researcher to generate annotation files, switch from their Python environment to a new software, and then only be able to browse and inspect the annotations. Moreover, because they were designed for annotation purposes, they do not provide advanced functionalities to easily locate hypothesis errors.

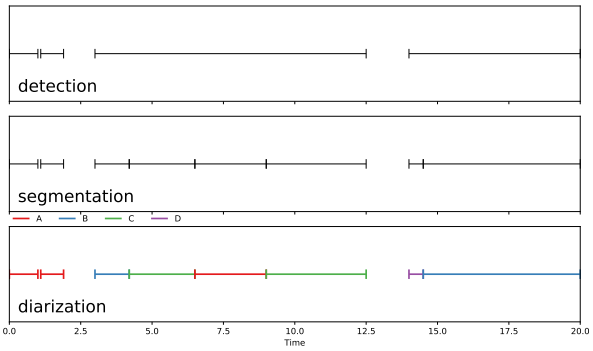

Figure 5: *Inspecting the output of speaker diarization modules*

Because it internally relies on `pyannote.core`, advanced visualization capabilities, `pyannote.metrics` addresses the first limitation. It can be used to visualize each step of any speaker diarization pipeline, without leaving the current Python or Jupyter Notebook [20] environment (see Appendix). Figure 5 was obtained using this feature – showing it can also be used to generate publication-quality figures.

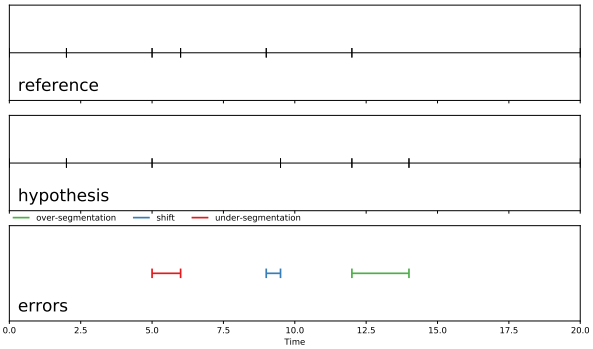

Figure 6: *Visualizing segmentation errors*

Finally, `pyannote.metrics` also provides a few tools dedicated to the analysis of segmentation or diarization errors. Figure 6 provides an example of segmentation error analysis. Given a reference and hypothesized segmentations, the third line is generated automatically and allows to quickly get insight at the type and location of segmentation errors. This example contains three errors: one boundary is not at the right location (*shift* error), one boundary is missing (*under-segmentation* error), and one segment is incorrecly split into two smaller ones (*over-segmentation* error).

5. Conclusion

In this paper, we introduced `pyannote.metrics` an open-source Python library for reproducible evaluation, diagnostic, and error analysis of speaker diarization systems. Installation instructions, example notebooks, and documentation can be found on the dedicated website: `http://pyannote.github.io`.

The list of implemented metrics is obviously not exhaustive (especially as far as error analysis is concerned) and `pyannote.metrics` is meant to be community-driven. Contributions are welcome: additional metrics, documentation improvement, new `pyannote.database` packages, ...

6. Appendix

Figure 7 showcases the integration with Jupyter Notebook, and provides a few examples of the `pyannote.metrics` object-oriented API.

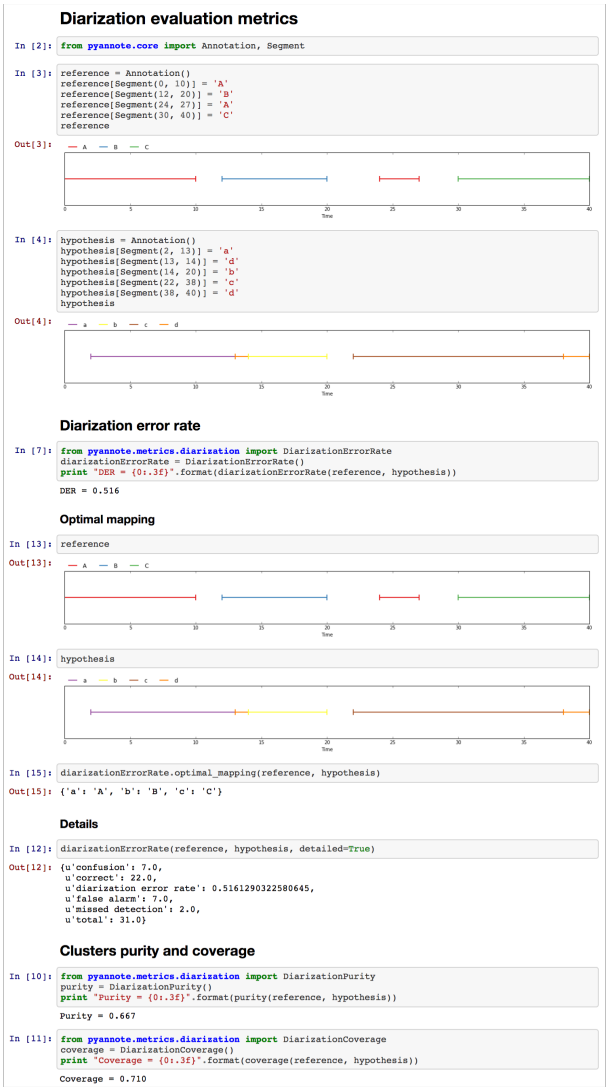

Figure 7: *Jupyter Notebook integration*

7. Acknowledgements

This work was partly supported by ANR through the ODESSA (ANR-15-CE39-0010) and MetaDaTV (ANR-14-CE24-0024) projects.

8. References

[1] J. G. Fiscus, N. Radde, J. S. Garofolo, A. Le, J. Ajot, and C. Laprun, *The Rich Transcription 2005 Spring Meeting Recognition Evaluation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 369–389.

[2] S. Galliano, E. Geoffrois, D. Mostefa, K. Choukri, J.-F. Bonastre, and G. Gravier, "The ESTER Phase II Evaluation Campaign for the Rich Transcription of French Broadcast News," in *Ninth Annual Conference of the International Speech Communication Association*, 2005.

[3] G. Gravier, G. Adda, N. Paulson, M. Carré, A. Giraudel, and O. Galibert, "The ETAPE Corpus for the Evaluation of Speech-based TV Content Processing in the French Language," in *LREC - Eighth international conference on Language Resources and Evaluation*, Turkey, 2012, p. na. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00712591

[4] NIST, "The rich transcription spring 2003 (rt-03s) evaluation plan," NIST, Gaithersburg, MD., Tech. Rep., Februrary 2003. [Online]. Available: http://www.nist.gov/speech/tests/rt/rt2003/spring/docs/rt03-springeval-plan-v4.pdf

[5] O. Galibert, "Methodologies for the evaluation of speaker diarization and automatic speech recognition in the presence of overlapping speech." in *Fourteenth Annual Conference of the International Speech Communication Association*, 2013.

[6] A. Larcher, K. A. Lee, and S. Meignier, "An extensible speaker identification SIDEKIT in Python," in *ICASSP 2016, IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2016.

[7] A. Anjos, L. E. Shafey, R. Wallace, M. Günther, C. McCool, and S. Marcel, "Bob: a free signal processing and machine learning toolbox for researchers," in *20th ACM Conference on Multimedia Systems (ACMMM), Nara, Japan*. ACM Press, Oct. 2012. [Online]. Available: https://publications.idiap.ch/downloads/papers/2012/Anjos_Bob_ACMMM12.pdf

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[10] X. Anguera, S. Bozonnet, N. Evans, C. Fredouille, G. Friedland, and O. Vinyals, "Speaker diarization: A review of recent research," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 2, pp. 356–370, 2012.

[11] A. Giraudel, M. Carr, V. Mapelli, J. Kahn, O. Galibert, and L. Quintard, "The REPERE corpus: a multimodal corpus for person recognition." in *LREC*, 2012, p. 11021107. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2012/pdf/707_Paper.pdf

[12] O. Galibert and J. Kahn, "The first official REPERE evaluation." in *SLAM 2013-First Workshop on Speech, Language and Audio for Multimedia*, 2013, p. 4348. [Online]. Available: http://ceur-ws.org/Vol-1012/papers/paper-08.pdf

[13] J. Carletta, "Unleashing the killer corpus: experiences in creating the multi-everything ami meeting corpus," *Language Resources and Evaluation*, vol. 41, no. 2, pp. 181–190, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10579-007-9040-x

[14] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[15] M. Cettolo, "Segmentation, classification and clustering of an Italian broadcast news corpus," in *Content-Based Multimedia Information Access-Volume 1*, 2000, pp. 372–381.

[16] J.-L. Gauvain, L. Lamel, and G. Adda, "Partitioning and transcription of broadcast news data." in *ICSLP 1998, 5th International Conference on Spoken Language Processing*, vol. 98, no. 5, 1998, pp. 1335–1338.

[17] C. Barras, X. Zhu, S. Meignier, and J.-L. Gauvain, "Multi-Stage Speaker Diarization of Broadcast News," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, no. 5, pp. 1505–1512, 2006.

[18] C. Barras, E. Geoffrois, Z. Wu, and M. Liberman, "Transcriber: Development and use of a tool for assisting speech corpora production," *Speech Communication*, vol. 33, no. 12, pp. 5 – 22, 2001.

[19] P. Boersma, "Praat, a system for doing phonetics by computer," *Glot International*, vol. 5:9/10, pp. 341 – 345.

[20] M. Ragan-Kelley, F. Perez, B. Granger, T. Kluyver, P. Ivanov, J. Frederic, and M. Bussonnier, "The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication." *AGU Fall Meeting Abstracts*, Dec. 2014.