SacreROUGE: An Open-Source Library for Using and Developing Summarization Evaluation Metrics

Daniel Deutsch and Dan Roth
Department of Computer and Information Science
University of Pennsylvania
{ddeutsch,danroth}@seas.upenn.edu

Abstract

We present SacreROUGE, an open-source library for using and developing summarization evaluation metrics.[1] SacreROUGE removes many obstacles that researchers face when using or developing metrics: (1) The library provides Python wrappers around the official implementations of existing evaluation metrics so they share a common, easy-to-use interface; (2) it provides functionality to evaluate how well any metric implemented in the library correlates to human-annotated judgments, so no additional code needs to be written for a new evaluation metric; and (3) it includes scripts for loading datasets that contain human judgments so they can easily be used for evaluation. This work describes the design of the library, including the core `Metric` interface, the command-line API for evaluating summarization models and metrics, and the scripts to load and reformat publicly available datasets. The development of SacreROUGE is ongoing and open to contributions from the community.

1 Introduction

Evaluating models is a critical step of the machine learning workflow. However, unlike classification-based tasks, evaluating models which generate text is difficult and is a research area on its own. The basic workflow for developing a new automatic evaluation metric is to design/implement the metric, calculate its correlation to human judgments, then use that metric to evaluate text generation systems.

While there have been significant efforts to build libraries for developing machine learning models (Klein et al., 2017; Gardner et al., 2018; Ott et al., 2019), no equivalent library exists for developing evaluation metrics. In this work, we present SacreROUGE, an open-source, Python-based library for using and developing text generation metrics, with an emphasis on summarization.

SacreROUGE removes many obstacles that researchers face when they use or develop evaluation metrics. First, the official implementations of various metrics do not share a common interface or programming language, so using many metrics to evaluate a model can be frustrating and time consuming. SacreROUGE provides Python-based wrappers around many evaluation metrics so they all implement a simple, easy-to-use interface regardless of how they are implemented internally (§2).

Second, evaluating metrics themselves can be tricky. Correlations between metric values and human judgments are calculated at several different granularities, there are multiple commonly used correlation coefficients, and fairly comparing human-written references to system output requires implementing jackknifing. Since the evaluation code in SacreROUGE is shared across all of the metrics, any metric which implements the common `Metric` interface can be evaluated without writing additional code (§3).

Third, datasets that contain judgments which are commonly used to evaluate metrics do not share the same format, so writing code to load each dataset requires writing a significant amount of effort. SacreROUGE provides scripts for popular summarization datasets that load and reformat them into a common schema so they can easily be used for evaluation (§4).

The development of SacreROUGE is ongoing. We intend to add more metrics and datasets to the library as they become available. Further, we encourage researchers to use the SacreROUGE framework to use existing metrics and develop new ones. SacreROUGE is released under the Apache 2.0 license and is open to contributions from the community.

[1] https://github.com/danieldeutsch/sacrerouge

2 The Metric Interface

The development of evaluation metrics for summarization has been an active area of research for two decades. However, the community has not converged on a consistent format for the input data, so each metric uses its own custom schema. Further, the published code for evaluation metrics is written in various programming languages based on which language was popular when the metric was proposed. These challenges make it very cumbersome to use multiple metrics to evaluate a summarization system. SacreROUGE addresses these two problems by unifying all of the metrics' implementations into a common interface called `Metric`. The interface provides a Pythonic API that allows for evaluating an individual summary or batch of summaries. Since all of the metrics share the same interface, evaluating a summarization system with several different metrics is trivial.

In order to support older evaluation metrics written in languages such as Perl or Java, we have written Python wrappers around the original code that still implement the `Metric` interface. Internally, the wrappers serialize the input summaries to the format required by the underlying metric, a subprocess is created to run the original metric's code, and the output is then loaded from disk again in Python. This way, we do not have to port the original metric's code to Python and end-users can still use the metrics with the Python API.

SacreROUGE currently supports the following evaluation metrics:

- AutoSummENG (Giannakopoulos et al., 2008)
- BERTScore (Zhang et al., 2019)
- BEwT-E (Tratz and Hovy, 2008)
- BLEURT (Sellam et al., 2020)
- METEOR (Denkowski and Lavie, 2014)
- MeMoG (Giannakopoulos and Karkaletsis, 2010)
- MoverScore (Zhao et al., 2019)
- NPowER (Giannakopoulos and Karkaletsis, 2013)
- Pyramid Score (Nenkova and Passonneau, 2004)
- PyrEval (Gao et al., 2019)
- QAEval (Deutsch et al., 2020)
- ROUGE (Lin, 2004), including a Python-port that we wrote, which is significantly faster than the original Perl version
- SIMetrix (Louis and Nenkova, 2009)
- SumQE (Xenouleas et al., 2019)

Among these metrics, 6 have original implementations in Java, 6 in Python, 1 in Perl, and 1 with no known official implementation (Pyramid Score).

Handling Dependencies Many of the evaluation metrics rely on external resources in the form of code, models, or data files. Setting up these dependencies in the right format to use the metrics can be difficult.

The SacreROUGE library addresses this problem by providing setup scripts for each metric which download or compile any required resources. To make this process as easy as possible for the end-user, these scripts are run through a `setup-metric` command. The command takes the name of the metric to setup, then downloads the required dependencies to a common folder which is managed by SacreROUGE. Abstracting the metric setup by a simple command makes it such that the end-user can quickly and easily begin using all of the metrics within the library.

While there is nothing technically to prevent SacreROUGE from being used in a Windows environment, thus far the scripts for handling the metrics' dependencies have been written for Linux-based systems.

3 Evaluating Systems and Metrics

The two most common use cases of an evaluation metric are to evaluate a summarization system and to evaluate a metric itself by calculating its correlation to human judgments. Since all of the metrics in SacreROUGE implement a common interface, the code for these procedures is shared, so developers of new metrics do not need to rewrite the code to implement these procedures. This logic is exposed through `evaluate`, `score`, and `correlate`, which are subcommands of `sacrerouge`, the entry point for the library's command-line interface.

The `evaluate` Subcommand The purpose of the `evaluate` subcommand is to calculate a metric's score for one summarization system on one dataset, which most typically occurs when researchers compare their system's performance to others'.

The `evaluate` subcommand accepts a specific metric and an input file that contains the output of a summarization system for an input corpus. The command will load the input data, pass it to the metric, and save the metric's output at the summary-level and system-level. The summary-level output contains the metric's value for each individual summary, whereas system-level output represents the average performance across the dataset and is most often reported in papers.

The `score` Subcommand The typical workflow for an evaluation of a metric is to first calculate the metric's score on a large number of summaries produced by multiple summarization systems on the same set of inputs. Then, a correlation coefficient is calculated between those scores and human judgments on the same set of summaries. The first step of this methodology is handled by the `score` subcommand.

The `score` subcommand is very similar to `evaluate` except for two key differences. First, the input data is not expected to be the output from a single system. Instead, it is expected to be the outputs from several summarization systems for the same sets of input documents. Providing the output from multiple systems at once allows SacreROUGE to score each of the summaries more efficiently than repeated calls to `evaluate` because it can avoid duplicate work, such as calculating reference summary-specific statistics.

Second, the `score` subcommand will run jackknifing on the input data when possible and necessary. Jackknifing is a procedure which allows the value of a metric on system-produced and human-written summaries to be fairly compared when the human-written summaries are used to assess the quality of the system summary. Briefly, if there is more than one reference summary, each reference is evaluated against all of the others. Each system summary is repeatedly evaluated against each possible subset of the reference summaries that has one reference removed. The final system summary score is an average across those evaluations. When jackknifing is performed, a `_jk` suffix is appended to the name of the metric which makes it clear that it is not comparable to the non-jackknifed version.

The `correlate` Subcommand After all of the summaries have been scored using the `score` subcommand, the second step of the meta-evaluation of a metric is to calculate the correlation of those scores to human judgments. This step is done via the `correlate` subcommand.

SacreROUGE calculates the three correlation coefficients most commonly used in summarization: Pearson, Spearman, and Kendall. Further, these correlations are computed at three different granularities: the summary-level, the system-level, and globally. The summary-level correlation calculates the average correlation per input. The system-level calculates the correlation between average system performances for each metric. The global correlation directly calculates the correlation between all of the observed metric values. The former two granularities are most often used in the summarization literature, and we refer the reader to Deutsch et al. (2020) for a more detailed description of how to calculate each of them.

Handling Different Input Requirements It is often the case that different metrics require different input data (e.g., some metrics use reference summaries, others need access to the input documents). Therefore, the required data must be loaded from the input file and the `evaluate` and `score` subcommands must pass the required data to the metric.

The interface for loading data from an input file in SacreROUGE is called a `DatasetReader`. For a given input file(s), a `DatasetReader` loads the `Fields` for the evaluation instances. A `Field` is a base class which contains the data for an input instance, such as a `DocumentsField` that maintains the contents of the input documents. Then, each evaluation instance contains a mapping from the name of a field to its data.

In order to pass the appropriate `Fields` to the summarization metrics, we require that every class that implements the `Metric` interface lists the names of the `Fields` that it uses. For instance, the wrapper for the document-based evaluation metric SIMetrix specifies it needs a field called `documents`, a key in the evaluation instance `Field` mapping. Then, once the input data has been loaded, the `evaluate` and `score` commands can pass the required data to a metric for evaluation.

Automatically Generated Subcommands It is desirable to have a different `evaluate` and `score` subcommand for each individual metric so that developers can easily specify different metric parameters on the command line. A naive implementation of this would require manually creating the subcommand for each metric. However, in order to eliminate as much boilerplate code as possible, SacreROUGE includes a feature to automatically generate these subcommands for any metric that implements the `Metric` interface.

Using Python's `inspect` and `typing` libraries, we are able to examine the constructor of each metric and generate a command-line argument for each parameter. For parameters with primitive types, the `argparse` library directly supports casting command line parameters to the correct types. However, some metrics may use complex types, such as a list of integers. In such situations, SacreROUGE assumes that the command line argument will be a string-serialized JSON object that can be deserialized into the required type at runtime. This allows us to automatically generate `evaluate` and `score` subcommands for every metric supported by the library.

4 A Common Dataset Format

Over the past two decades, the summarization community has collected a large number of summarization datasets and human quality annotations. However, these very useful datasets are seldom saved in a common format, forcing every researcher who wants to train a model on the datasets or use the judgments to evaluate a metric to write boilerplate code to load the data.

To mitigate this issue, SacreROUGE provides scripts that will load the datasets and their corresponding judgments, then serialize them to new files with a common format. The data is serialized in such a manner that it can be directly used in the `evaluate`, `score`, and `correlate` subcommands, thereby making it incredibly easy to run or evaluate any metric in the library on the dataset.

The preprocessing scripts will save the data in a human-readable form into three different JSONL files:[2] one for the summarization task data, one for the summaries that have been scored by the human judges, and one for the corresponding metric scores for those summaries. Each object in the task file is uniquely identified by an instance ID and contains the input documents and ground-truth summaries for one instance. The scored summaries and metric scores files have parallel data. The objects in both files contain an instance ID that identifies the input documents that were used to generate the summary, a summarizer ID that identifies the summarization model used to produce the summary, and a string that identifies the type of summarization (either model-generated or human-written). The summaries file contains the actual summary output by the system and the references for the input instance, and the metrics file contains the corresponding metric scores for that summary. Example JSON objects from all of the files can be seen in Appendix A

The scripts to preprocess the datasets are exposed through the `setup-dataset` subcommand. The subcommand accepts the name of a dataset, an output directory, and any potential dataset-specific arguments. Then, SacreROUGE will load and preprocess the respective dataset. For datasets which are publicly available, the scripts will download the data automatically. However, many summarization datasets are licensed, so the corresponding preprocessing scripts require paths to the original data supplied to the command.

The datasets which are currently supported by SacreROUGE are the Document Understanding Conference from 2001 to 2007,[3] Text Analysis Conference from 2008 to 2011,[4] the MultiLing 2011, 2013, 2015, 2017, and 2019 Workshops,[5] and the CNN/DailyMail dataset judgments provided by Chaganty et al. (2018) and Fabbri et al. (2020). We intend to add more datasets as they become available, and other researchers can easily incorporate their own datasets to our library by serializing the data into the shared format.

5 Related Work

The idea for SacreROUGE came from the SacreBLEU (Post, 2018) library. SacreBLEU was developed to standardize and simplify calculating BLEU (Papineni et al., 2002) for machine translation. Like SacreROUGE, it provides a simple command-line interface to download and evaluate on common machine translation datasets. Whereas SacreBLEU is mainly for evaluating machine translation models with BLEU, our library focuses on summarization

[2]A JSONL file contains one serialized JSON per line.

[3]`https://duc.nist.gov/`

[4]`https://tac.nist.gov/`

[5]`http://multiling.iit.demokritos.gr/`

and includes a large number of evaluation metrics. Further, SacreROUGE also provides a framework for developing and evaluating new metrics.

One goal of SacreROUGE is to standardize the implementation and meta-evaluation of metrics. The Perl-based Asiya (Giménez and Màrquez, 2010) and Python-based EASSE (Alva-Manchego et al., 2019) have been developed with similar goals for machine translation and text simplification, respectively.

Much of the design of SacreROUGE was inspired by AllenNLP (Gardner et al., 2018), a library built on PyTorch (Paszke et al., 2017) for developing deep learning models. AllenNLP provides useful abstractions over different models and neural network modules that allows for the sharing of boilerplate code so developers can quickly create and train new machine learning models. SacreROUGE provides similar abstractions for evaluation metrics.

Two concurrent works set out to achieve similar goals to SacreROUGE, the `nlp` library from Hugging Face[6] and SummEval (Fabbri et al., 2020). Both libraries provide easy-to-use interfaces for running many evaluation metrics. However, SacreROUGE provides much more functionality for developing and evaluating new metrics, including providing dataset readers for benchmark metric evaluation datasets, implementations of features such as jackknifing, and calculating common correlation coefficients.

6 Conclusion

We have presented SacreROUGE, an open-source library dedicated to the development of summarization evaluation metrics. With a unified metric interface and common data format, our library makes it very simple to use existing evaluation metrics as well as develop new ones with a minimum amount of effort. We hope that future researchers will contribute their own metrics and datasets to the library so that it is as easy as possible to run and evaluate summarization metrics.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful suggestions and feedback.

This work was supported by Contract FA8750-19-2-1004 and Contract FA8750-19-2-0201 with the US Defense Advanced Research Projects Agency (DARPA). Approved for Public Release, Distribution Unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

This research is supported by a Focused Award from Google.

[6]https://github.com/huggingface/nlp

References

Fernando Emilio Alva-Manchego, Louis Martin, Carolina Scarton, and Lucia Specia. 2019. EASSE: easier automatic sentence simplification evaluation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019 - System Demonstrations*, pages 49–54. Association for Computational Linguistics.

Arun Chaganty, Stephen Mussmann, and Percy Liang. 2018. The price of debiasing automatic metrics in natural language evalaution. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 643–653, Melbourne, Australia. Association for Computational Linguistics.

Michael J. Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation, WMT@ACL 2014, June 26-27, 2014, Baltimore, Maryland, USA*, pages 376–380. The Association for Computer Linguistics.

Daniel Deutsch, Tania Bedrax-Weiss, and Dan Roth. 2020. Towards Question-Answering as an Automatic Metric for Evaluating the Content Quality of a Summary.

Alexander R Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. 2020. SummEval: Re-evaluating Summarization Evaluation. *arXiv preprint arXiv:2007.12626*.

Yanjun Gao, Chen Sun, and Rebecca J. Passonneau. 2019. Automated pyramid summarization evaluation. In *Proceedings of the 23rd Conference on Computational Natural Language Learning, CoNLL 2019, Hong Kong, China, November 3-4, 2019*, pages 404–418. Association for Computational Linguistics.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for*

NLP Open Source Software (NLP-OSS), pages 1–6, Melbourne, Australia. Association for Computational Linguistics.

George Giannakopoulos and Vangelis Karkaletsis. 2010. Summarization system evaluation variations based on n-gram graphs. In *Proceedings of the Third Text Analysis Conference, TAC 2010, Gaithersburg, Maryland, USA, November 15-16, 2010*. NIST.

George Giannakopoulos and Vangelis Karkaletsis. 2013. Summary evaluation: Together we stand npower-ed. In *Computational Linguistics and Intelligent Text Processing - 14th International Conference, CICLing 2013, Samos, Greece, March 24-30, 2013, Proceedings, Part II*, volume 7817 of *Lecture Notes in Computer Science*, pages 436–450. Springer.

George Giannakopoulos, Vangelis Karkaletsis, George A. Vouros, and Panagiotis Stamatopoulos. 2008. Summarization system evaluation revisited: N-gram graphs. *TSLP*, 5(3):5:1–5:39.

Jesús Giménez and Lluís Màrquez. 2010. Asiya: An open toolkit for automatic machine translation (meta-)evaluation. *Prague Bull. Math. Linguistics*, 94:77–86.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Annie Louis and Ani Nenkova. 2009. Automatically evaluating content selection in summarization without human models. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7 August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 306–314. ACL.

Ani Nenkova and Rebecca J. Passonneau. 2004. Evaluating content selection in summarization: The pyramid method. In *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2004, Boston, Massachusetts, USA, May 2-7, 2004*, pages 145–152. The Association for Computational Linguistics.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*, pages 311–318. ACL.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Thibault Sellam, Dipanjan Das, and Ankur P. Parikh. 2020. BLEURT: Learning Robust Metrics for Text Generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7881–7892. Association for Computational Linguistics.

Stephen Tratz and Eduard H. Hovy. 2008. Summarization evaluation using transformed basic elements. In *Proceedings of the First Text Analysis Conference, TAC 2008, Gaithersburg, Maryland, USA, November 17-19, 2008*. NIST.

Stratos Xenouleas, Prodromos Malakasiotis, Marianna Apidianaki, and Ion Androutsopoulos. 2019. SUM-QE: a bert-based summary quality estimation model. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 6004–6010. Association for Computational Linguistics.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with BERT. *CoRR*, abs/1904.09675.

Wei Zhao, Maxime Peyrard, Fei Liu, Yang Gao, Christian M. Meyer, and Steffen Eger. 2019. Moverscore: Text generation evaluating with contextualized embeddings and earth mover distance. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 563–578. Association for Computational Linguistics.

A Example Dataset Files

Example JSON objects that are serialized in the task, scored summary, and metric scores files (see §4) are included in Listings 1, 2, and 3.

```
{
  "instance_id": "d0801-A",
  "documents": [
    {
      "filename": "AFP_ENG_20050115.0485",
      "text": "The A380, the new Airbus..."
    },
    ...
  ],
  "summaries": [
    {
      "annotator": "G",
      "text": "In 1994 Airbus began engineering..."
    },
    ...
  ]
}
```

Listing 1: An example JSON object included in the files that contain the summarization task data. The `documents` and `summaries` objects are the list of input documents and reference summaries for the instance identified by `instance_id`.

```
{
  "instance_id": "d0801-A",
  "summarizer_id": "0",
  "summarizer_type": "peer",
  "summary": {
    "text": "The superjumbo Airbus A380..."
  }
  "references": [
    {
      "summarizer_id": "G",
      "text": "In 1994 Airbus began engineering..."
    },
    ...
  ]
}
```

Listing 2: An example JSON object included in the files that contain the summaries which were scored by human judges. The `summary` object is the summary output by the system identified by `summarizer_id` on input instance `instance_id`. The `summarizer_type` field marks the summary was produced by a model (also called a "peer") and not a human. The `references` object contains the reference summaries for this input instance.

```
{
  "instance_id": "d0801-A",
  "summarizer_id": "0",
  "summarizer_type": "peer",
  "metrics": {
    "overall_responsiveness": 3,
    "modified_pyramid_score": 0.241,
    "rouge-1": {
      "recall": 26.7,
      "precision": 29.4,
      "f1": 28.0
    },
    ...
  }
}
```

Listing 3: An example JSON object included in the files that contain the metric values for the scored summaries. The `instance_id` and `summarizer_id` identify that these metrics are for the the summary data included in the parallel file (i.e., Listing 2). That summary's metric values are contained in the `metrics` dictionary.