CEUR-WS.org/Vol-3164/paper30.pdf

A Framework for Automated Text Generation Benchmarking

Steven Layne[1,2], Sebastian Gehrmann[3], Franck Dernoncourt[1], Lidan Wang[1], Trung Bui[1] and Walter Chang[1]

[1]*Adobe Research*
[2]*University of Illinois Urbana Champaign*
[3]*Harvard University*

Abstract

Researchers in areas such as translation and summarization need to compare their results to a wide range of published baselines that commonly use different evaluation methods. We aim to enable an easy comparison by presenting TextGen-Benchmarch, an open-sourced tool[1] for streamlining the generation and evaluation of text. Text generation methods and evaluation metrics can easily be added to TextGen-Benchmarch, and its pipeline results in a more efficient comparison between methods as users can supply corpora, systems, and evaluation techniques and receive comparison reports in easy to analyze tabular and graphic formats.

Keywords

Summarization, Text generation, evaluation

1. Introduction

An in-depth evaluation and a fair comparison to the current literature are crucial parts in the development of Machine Learning (ML) systems. In addition to model-specific investigations, this evaluation process typically includes automated metrics that allow predictions to be compared to those of other approaches. However, subtle differences in output formatting or evaluation metrics can lead to drastically different reported results [1]. It is thus of particular importance to ensure a homogeneous evaluation environment that applies the same evaluation to each system output.

In the case of (conditional) text generation problems, the goal is to generate text that is conditioned on an input and subject to constraints defined by the task, for example, the length. Depending on the task, there are various metrics that can be applied for the evaluation, such as ROUGE [2, 3], METEOR [4], BLEU [5], NIST [6], or CIDEr [7]. A commonality between these metrics is that all of them compare a generated text against one or many, typically human-generated, references. These references are a demonstration of what an adequate result should look like for a given task and input. The metrics are used to give a quantitative measure of quality for generated text with respect to the reference(s). However, every metric uses different input and output formats. Moreover, some metrics like ROUGE and METEOR can be configured with multiple parameters. For example, the α parameter in ROUGE mediates the preference for precision or recall for computing F-Measures [8]. Any change in the selection of options results in a different result and the evaluation options are typically not reported along with published results. Thus, it is necessary to compare outputs of multiple systems with the same options across multiple metrics to ensure a fair comparison.

We propose TextGen-Benchmarch, which simplifies the process of model evaluation by streamlining the benchmarking process and enabling the quick comparison of text-generation systems for a given task. The framework is agnostic to the underlying problem and implements a wide range of common evaluation metrics. Moreover, TextGen-Benchmarch provides a simple API to include additional models. During the evaluation, it can use either cached or user-provided predictions or use the model API to run inference on a given sample. We demonstrate the effectiveness of the tool for the problem of extractive summarization and show how it can make a comparison between related approaches easier and more well-rounded.

2. Related Work

Some tools encapsulate different metrics into a single library so that users can evaluate their hypotheses against references using a shared interface.While these tools suc-

The AAAI 2022 Workshop on Scientific Document Understanding
stevenlayne2017@u.northwestern.edu (S. Layne); gehrmann@seas.harvard.edu (S. Gehrmann); franck.dernoncourt@gmail.com (F. Dernoncourt); lidwang@adobe.com (L. Wang); bui@adobe.com (T. Bui); wachang@adobe.com (W. Chang)
http://francky.me/ (F. Dernoncourt)
0000-0002-8257-9516 (S. Gehrmann); 0000-0002-1119-1346 (F. Dernoncourt)
© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

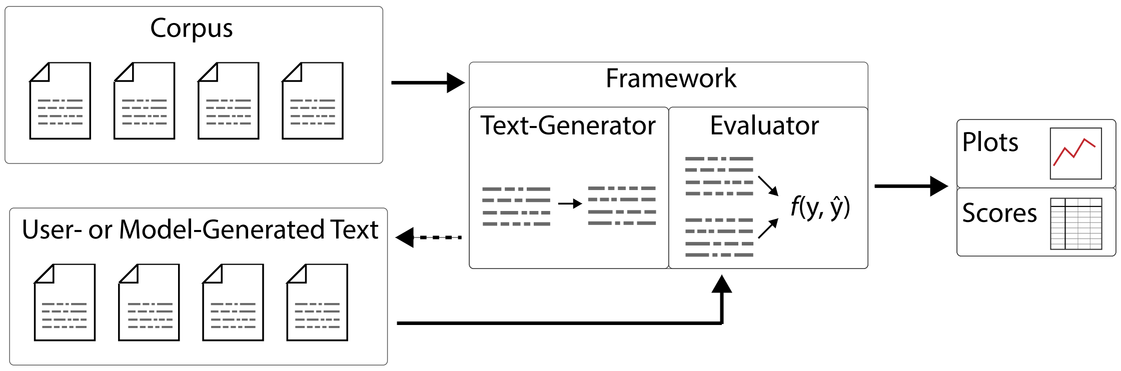

Figure 1: Demonstration of the TextGen-Benchmarch pipeline. The framework uses the Evaluator to compare gold-references against predictions and generates plots and tables to summarize the results. The predictions can either take the form of predefined user-input, cached outputs from previous runs, or generated with the Text-Generator module that can be extended with any model.

cessfully enable the evaluation of a specific system, they are limited to a single system at a time. Therefore, each user is required to develop their own comparison.

Some libraries are also restricted in the compatible input formats. For example, the COCO (Common Objects in Context) Caption evaluation library [9] provides an interface that was created to evaluate captioning results. It has support for BLEU, METEOR, ROUGE-L, CIDEr, and SPICE. The evaluation library enables users of COCO caption to streamline the evaluation of their results but is limited to COCO-compatible input objects as the library was intended to be used in the context of the MS-COCO Evaluation Server [9].

Other libraries can compare models with different input formats, but only for limited tasks. For example, Spark provides ML Pipelines [1], a high-level API for their data handlers. At the end of a pipeline, users may pass their results to evaluators which are designed for classification and regression models, and do not serve text generation models.

3. System Overview

The TextGen-Benchmarch framework is built in Python and provides a pipeline as illustrated in Figure 1. Before starting, TextGen-Benchmarch parses a configuration file that contains (1) the paths to datasets, (2) the systems, and (3) the metrics to be used. It additionally allows for descriptors for the text format. For example, if sentences are surrounded by tags that should be ignored during evaluation, it can be specified here. Any specified dataset must contain two sub-folders *samples* and *gold*. The *gold* folder contains files with line separated references.[2] Samples are read in using Python's file-stream which ensures minimal memory usage. The references can be stored as either plain-text or as a JSON file to enable multiple references. Here, each line should be formatted as follows:

```
{"references" :
    ["ref 1","ref 2", "ref 3"]
}
```

TextGen-Benchmarch loads samples from the datasets specified in the configuration file. It parses the files and passes one document at a time to the Text-Generator. The Text-Generator returns model-generated text, which is then stored in the file system to be used during evaluation. Users may provide their own generated text in conjunction with model-generated texts or skip text generation entirely by turning off the generation in configuration. Text generation is also skipped if TextGen-Benchmarch infers that a given dataset has already been processed with the model and is cached on the file system. If the evaluation is enabled, the user and model-generated text are evaluated against the reference texts. TextGen-Benchmarch currently supports ROUGE, METEOR, NIST, and BLEU scores. We provide additional details on how the library interfaces with data in Section 4.

4. Extending the System

TextGen-Benchmarch is designed to make it as easy as possible for users to add and remove text generators and metrics. TextGen-Benchmarch interfaces with two

[1]https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html

[2]Python natively supports file-stream with line separated files which is why it is a formatting requirement.

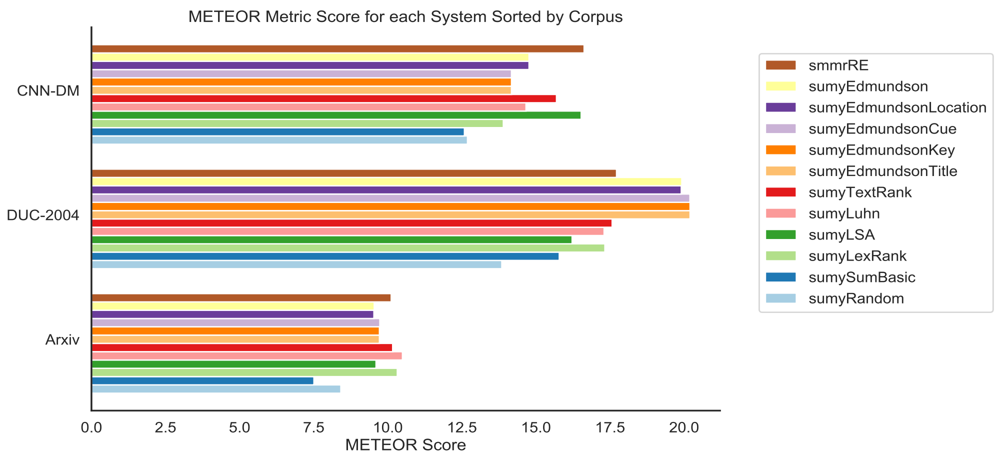

Figure 2: Meteor Score Report on DUC, ArXiv, and CNN-DM

library files – one for metrics and one for text generators. Additions can be added to these two libraries.

4.1. Adding text generators

The text generator library provides a single public method with two inputs: a targeted text generator and text. The targeted text-generator is called and it returns the resulting text. A user can add additional models by adding a method for their model that takes in text as input and returns a generated text as output.

On load of the library class, information related to the format of samples is saved. This information includes separators for tokenized sentences and a Boolean that indicates whether the text is tokenized. Custom methods must use this information to decide how to preprocess the input text before passing it into the flow of their added model. Some text generators require sentences to be pre-tokenized whereas other text generators have custom tokenizers and expect raw text. For additional convenience, we provide an interface for a tokenizer and a detokenizer with the library.

4.2. Adding Metrics

Adding metrics follows a similar process to to one outlined for text generators. The metric library provides a method with a single input: a custom Summary Reader Object (SRO). The SRO has two public methods: `readOne` and `readAll`. When `readOne` is called a tuple of the form `(prediction, references)`. When `readAll` is called, a list of all *N* tuples is returned, where *N* corresponds to the number of generated texts.

The `readOne` and `readAll` methods are abstractions for Python's file-stream reader.

5. Report types

TextGen-Benchmarch provides the following report types.

- *CSV: Fixed Metric* generates a separate CSV for each metric. Each row is a different model. Each column represents a corpus. To assist with comparisons of the same evaluation metric and set of summarizers but against different corpora.
- *CSV: Fixed corpus* generates a single report for one corpus. Each row represents a model and each column a metric. This assists with comparisons on the same corpus with a single set of models but against different metrics.
- *Horizontal Barchart: Fixed Metric.* Grouped by the corpus, this shows scores on the X-axis, sorted by average metric score across corpora. This visualization helps draw comparisons between models across different corpora.

6. Example Reports

We demonstrate the usage of TextGen-Benchmarch using the extractive summarization problem. An extractive summary is defined as a subset of sentences from a number of documents (either one or many) that effectively summarizes the message of the input. Typical metrics for this task include ROUGE and METEOR. We present a comparison of popular non-parametric extractive summarizers on the DUC 2004 [10], ArXiv [11] and CNN-DM [12, 13] datasets respectively. We are comparing smmrRE, our re-implementation of SMMRY extractive summarizer [3], and Python's sumy summarizers [4].

For ArXiv and CNN-DM, we used 1,000 samples of the test-set for demonstration purposes. Thus, the results should not be interpreted as official scores. They do, however, highlight some interesting variation between the performance of the summarizers in the different metrics.

Figure 2 shows the METEOR scores. The order corresponds, from top to bottom, to a summarizer's rank when comparing the average score across all corpora. Here, smmrRE ranks first and sumyRandom comes in last.

References

[1] S. A. Ellafi, Preprocessing and normalization for automatic evaluation of machine translation, Association for Computational Linguistics, 2005.

[2] C.-Y. Lin, E. Hovy, Automatic evaluation of summaries using n-gram co-occurrence statistics, in: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1, Association for Computational Linguistics, 2003, pp. 71–78.

[3] C.-Y. Lin, ROUGE: A package for automatic evaluation of summaries, in: Text summarization branches out: Proceedings of the Association for Computational Linguistic workshop, volume 8, 2004.

[4] S. Banerjee, A. Lavie, METEOR: An automatic metric for mt evaluation with improved correlation with human judgments, in: Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization, volume 29, 2005, pp. 65–72.

[5] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu, Bleu: a method for automatic evaluation of machine translation, in: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, 2002. URL: http://aclweb.org/anthology/P02-1040.

[6] G. Doddington, Automatic evaluation of machine translation quality using n-gram co-occurrence statistics, 2002, pp. 138–145.

[7] R. Vedantam, C. Lawrence Zitnick, D. Parikh, Cider: Consensus-based image description evaluation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 4566–4575.

[8] C.-Y. Lin, A brief introduction of the rouge summary evaluation package, Univeristy of Southern California/Information Sciences Institute, 2005.

[9] X. Chen, H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollár, C. L. Zitnick, Microsoft coco captions: Data collection and evaluation server, CoRR abs/1504.00325 (2015).

[10] P. Over, J. Yen, Introduction to DUC-2004: an intrinsic evaluation of generic news text summarization systems, 2004.

[11] A. Cohan, F. Dernoncourt, D. S. Kim, T. Bui, S. Kim, W. Chang, N. Goharian, A discourse-aware attention model for abstractive summarization of long documents, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), Association for Computational Linguistics, 2018, pp. 615–621. URL: http://aclweb.org/anthology/N18-2097. doi:10.18653/v1/N18-2097.

[12] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, P. Blunsom, Teaching machines to read and comprehend, in: Advances in Neural Information Processing Systems, 2015, pp. 1693–1701.

[13] R. Nallapati, B. Zhou, C. dos Santos, C. Gulcehre, B. Xiang, Abstractive text summarization using sequence-to-sequence rnns and beyond, in: Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning, Association for Computational Linguistics, 2016, pp. 280–290. URL: http://aclweb.org/anthology/K16-1028. doi:10.18653/v1/K16-1028.

[3]https://smmry.com/about
[4]https://github.com/miso-belica/sumy