Sign up Sign In

Search Medium Write

Published in Towards Data Science

You have **2** free member-only stories left this month.
Sign up for Medium and get an extra one

James Briggs Follow

Mar 4, 2021 · 6 min read · Member-only · Listen

The Ultimate Performance Metric in NLP

Never worry about accuracy in your language models again

231 2

Photo by Wynand van Poortvliet on Unsplash

Measuring the results of our model outputs gets a lot more complex when we're dealing with language.

This is something that becomes quite clear very quickly for many NLP-based problems — how do we measure the accuracy of a language-based sequence when dealing with language summarization or translation?

For this, we can use Recall-Oriented Understudy for Gisting Evaluation (ROUGE). Fortunately, the name is deceptively complicated — it's incredibly easy to understand, and even easier to implement.

Let's jump straight into it.

Contents

What is ROUGE

ROUGE is actually a set of metrics, rather than just one. We will cover the main ones that are most likely to be used, starting with ROUGE-N.

ROUGE-N

ROUGE-N measures the number of matching 'n-grams' between our model-generated text and a 'reference'.

An n-gram is simply a grouping of tokens/words. A unigram (1-gram) would consist of a single word. A bigram (2-gram) consists of two consecutive words:

Original: "*the quick brown fox jumps over*"

Unigrams: *['the', 'quick', 'brown', 'fox', 'jumps', 'over']*

Bigrams: *['the quick', 'quick brown', 'brown fox', 'fox jumps', 'jumps over']*

Trigrams: *['the quick brown', 'quick brown fox', 'brown fox jumps', 'fox jumps over']*

ngrams.md hosted with ❤ by **GitHub** **view raw**

The reference is a human-made best-case output — so for automated summarization is would be a human-made summary of our input text. For machine translation, it would be a professional translation of our input text.

With ROUGE-N, the N represents the n-gram that we are using. For ROUGE-1 we would be measuring the match-rate of unigrams between our model output and reference.

ROUGE-2 and ROUGE-3 would use bigrams and trigrams respectively.

Once we have decided which N to use — we now decide on whether we'd like to calculate the ROUGE recall, precision, or F1 score.

Recall

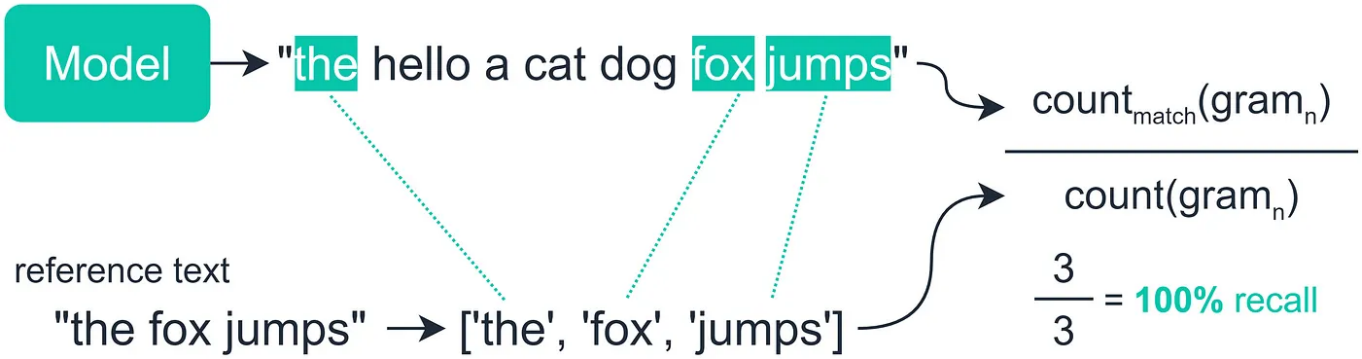
The **recall** counts the number of overlapping n-grams found in both the model output and reference — then divides this number by the total number of n-grams in the reference. It looks like this:

$$\frac{\text{number of n-grams found in model and reference}}{\text{number of n-grams in reference}}$$

$$\frac{\text{count}_{match}(\text{gram}_n)}{\text{count}(\text{gram}_n)}$$

The calculation of our ROUGE-N recall metric for a single sample, in plain English (top) and simplified notation (bottom)

This is great for ensuring our model is **capturing all of the information** contained in the reference — but this isn't so great at ensuring our model isn't just pushing out a huge number of words to game the recall score:



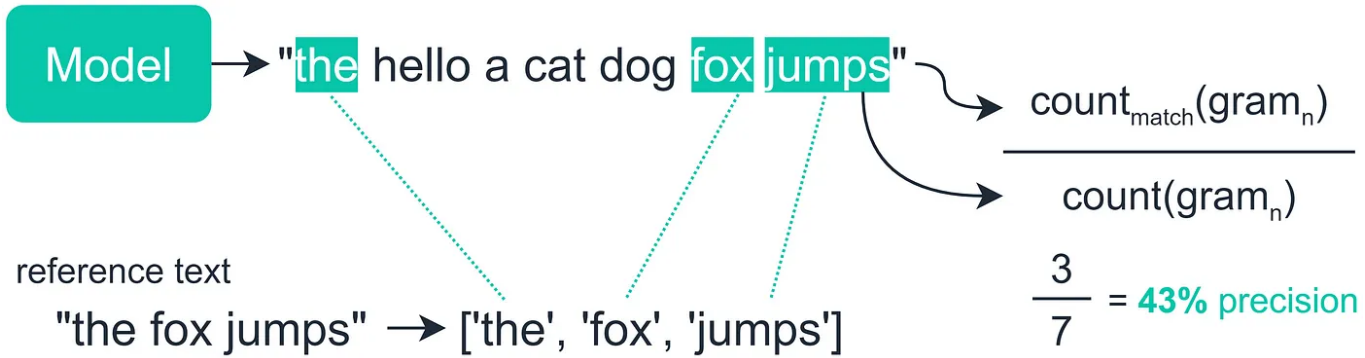
Our model could just output every single word in our vocabulary and get a **perfect** recall score every time without fail

Precision

To avoid this we use the **precision** metric — which is calculated in almost the exact same way, but rather than dividing by the **reference** n-gram count, we divide by the **model** n-gram count.

$$\frac{\text{number of n-grams found in model and reference}}{\text{number of n-grams in } \mathbf{model}}$$

So if we apply this to our previous example, we get a precision score of just **43%**:

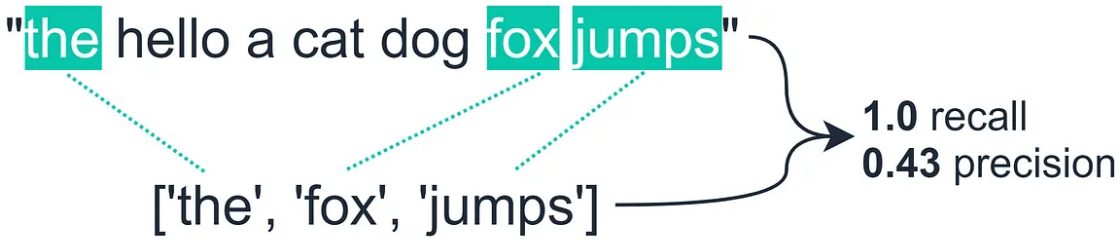


F1-Score

Now that we both the recall and precision values, we can use them to calculate our ROUGE F1 score like so:

$$2 * \frac{precision * recall}{precision + recall}$$

Let's apply that again to our previous example:



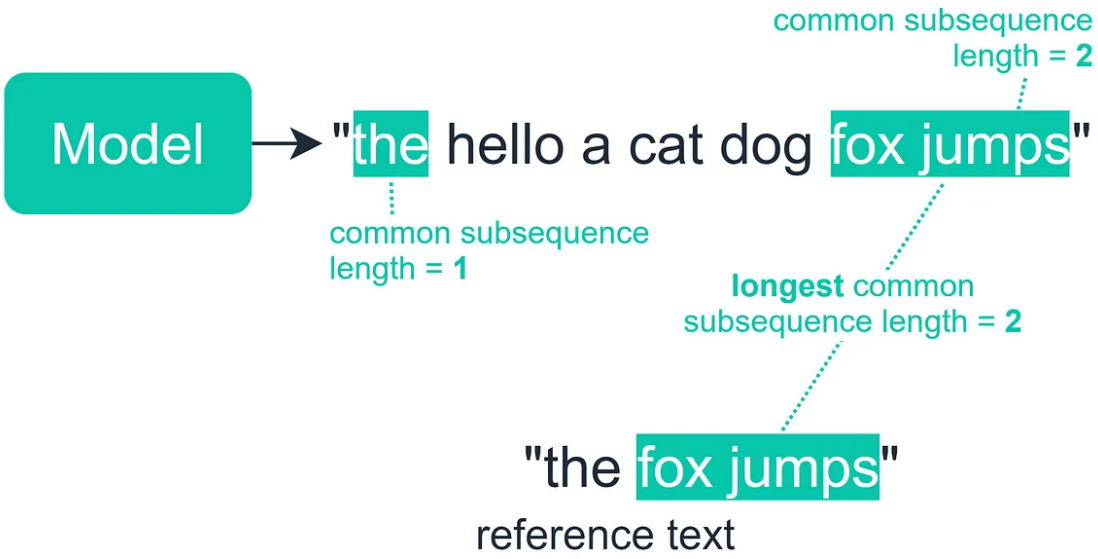
$$2 * \frac{0.43 * 1.0}{0.43 + 1.0} = 0.6$$

60% f1 score

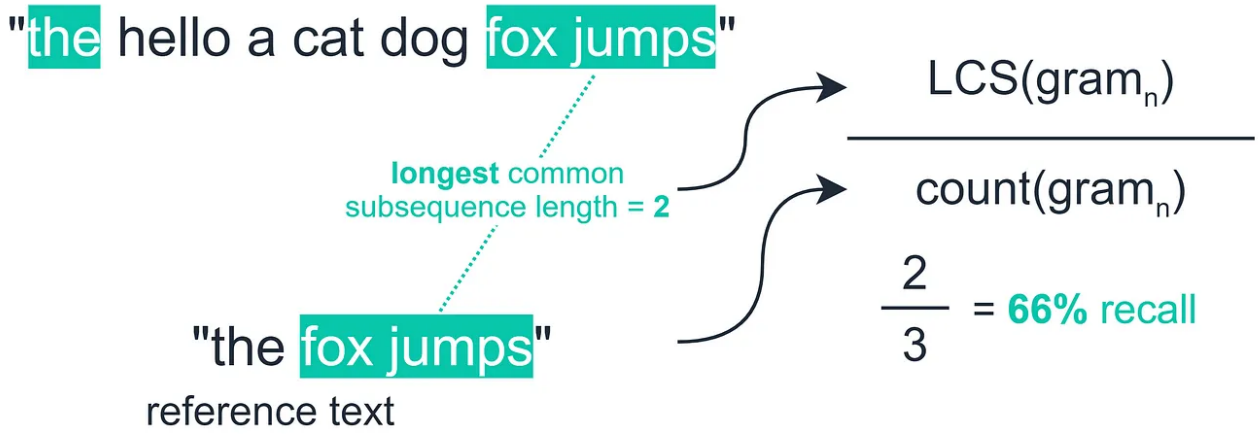
That gives us a reliable measure of our model performance that relies not only on the model capturing as many words as possible (recall) but doing so without outputting irrelevant words (precision).

ROUGE-L

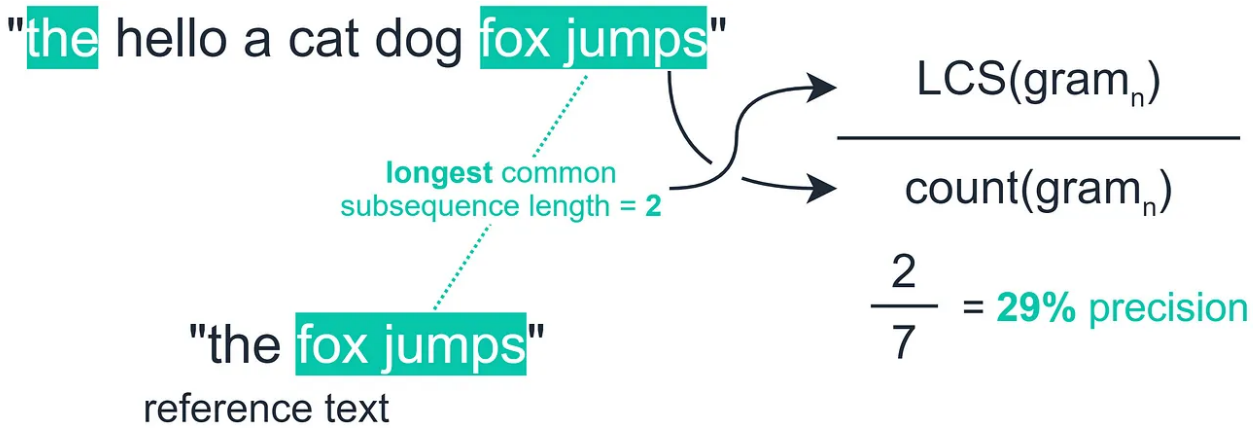
ROUGE-L measures the longest common subsequence (LCS) between our model output and reference. All this means is that we count the longest sequence of tokens that is shared between both:



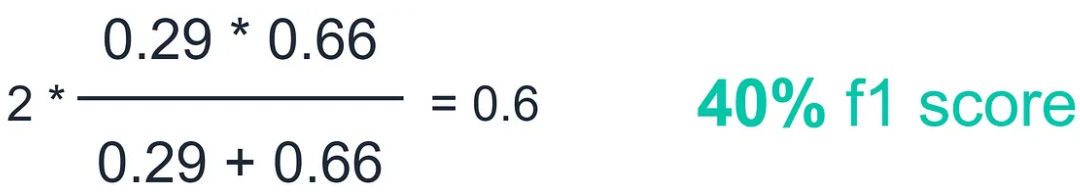
The idea here is that a longer shared sequence would indicate more similarity between the two sequences. We can apply our recall and precision calculations just like before — but this time we replace the **match** with **LCS:**



Our LCS recall calculation



Precision is much the same but we switch our total n-gram count from the reference to the model



And finally, we calculate the F1 score just like we did before

ROUGE-S

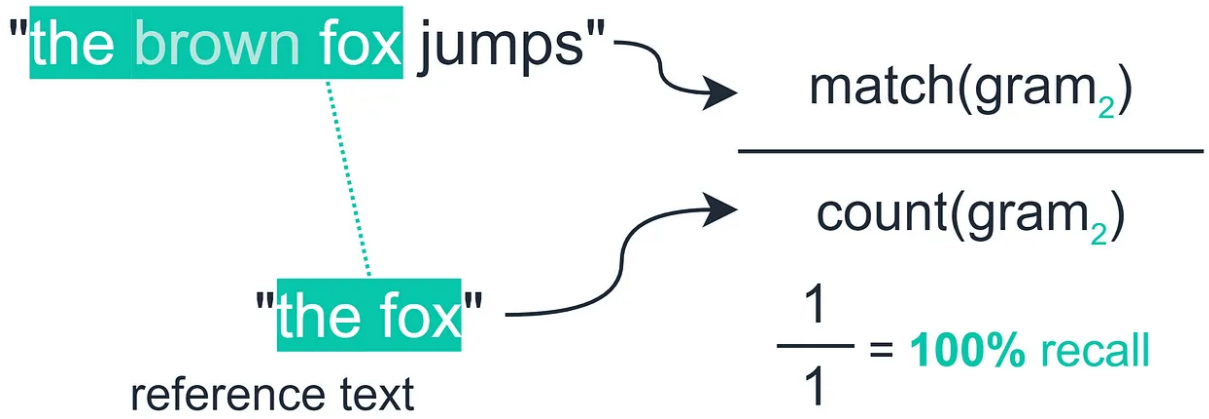
The final ROUGE metric we will look at is the ROUGE-S — or skip-gram concurrence metric.

Now, this metric seems to be much less popular than ROUGE-N and ROUGE-L covered already — but it's worth being aware of what it does.

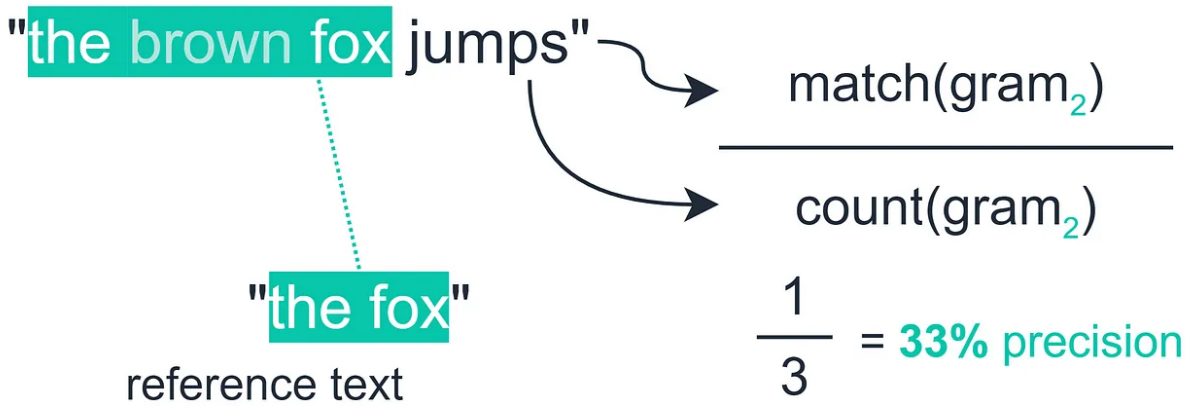
Using the skip-gram metric allows us to search for consecutive words from the reference text, that appear in the model output but are separated by one-or-more other words.

So, if we took the bigram “the fox”, our original ROUGE-2 metric would only match this if this exact sequence was found in the model output. If the model instead outputs “the brown fox” — no match would be found.

ROUGE-S allows us to add a degree of leniency to our n-gram matching. For our bigram example we could match by using a skip-**bi**gram measure:



We calculate recall just like we did with ROUGE-N — but we add in leniency for any words appearing between matches



The same applies to our precision metric too

After calculating our recall and precision, we can calculate the F1 score too just as we did before.

Cons

ROUGE is a great evaluation metric but comes with some drawbacks. In-particular, ROUGE does not cater for different words that have the same meaning — as it measures syntactical matches rather than semantics.

So, if we had two sequences that had the same meaning — but used different words to express that meaning — they could be assigned a low ROUGE score.

This can be offset slightly by using several references and taking the average score, but this will not solve the problem entirely.

Nonetheless, it's a good metric for assessing both machine translation and automatic summarization tasks and is very popular for both.

In Python

Fortunately, implementing these metrics in Python is incredibly easy thanks to the Python rouge library.

We can install the library through pip:

```
pip install rouge
```

And scoring our model output against a reference is as easy as this:

```
In [1]: from rouge import Rouge

In [2]: model_out = "he began by starting a five person war cabinet and included chambe

        reference = "he began his premiership by forming a five-man war cabinet which i

In [3]: rouge = Rouge()

In [4]: rouge.get_scores(model_out, reference)

[   {   'rouge-1': {   'f': 0.7567567517604091,
                       'p': 0.7777777777777778,
                       'r': 0.7368421052631579},
        'rouge-2': {'f': 0.514285709289796, 'p': 0.5294117647058824, 'r': 0.5},
        'rouge-l': {   'f': 0.7567567517604091,
                       'p': 0.7777777777777778,
                       'r': 0.7368421052631579}}]
```

rouge.ipynb hosted with ❤ by GitHub view raw

The `get_scores` method returns three metrics, ROUGE-N using a unigram (ROUGE-1) and a bigram (ROUGE-2) — and ROUGE-L.

For each of these, we receive the F1 score `f`, precision `p`, and recall `r`.

For Datasets

Typically we would be calculating these metrics for a set of predictions and references — to do this we format our predictions and references into a list of predictions and references respectively — then we add the `avg=True` argument to `get_scores` like so:

```
In [1]: from rouge import Rouge

In [2]: model_out = ["he began by starting a five person war cabinet and included chamb
                     "the siege lasted from 250 to 241 bc, the romans laid siege to li
                     "the original ocean water was found in aquaculture"]

        reference = ["he began his premiership by forming a five-man war cabinet which
                     "the siege of lilybaeum lasted from 250 to 241 bc, as the roman ar
                     "the original mission was for research into the uses of deep ocear

In [3]: rouge = Rouge()

In [4]: rouge.get_scores(model_out, reference, avg=True)

{   'rouge-1': {   'f': 0.6279006234427593,
                   'p': 0.8604497354497355,
                   'r': 0.5273531655225019},
    'rouge-2': {   'f': 0.3883256484545606,
                   'p': 0.5244559362206421,
                   'r': 0.32954545454545453},
    'rouge-l': {   'f': 0.6282785202429159,
                   'p': 0.8122895622895623,
                   'r': 0.5369305616983636}}
```

rouge_avg.ipynb hosted with ❤ by GitHub view raw

That's all for this article on understanding and implementing the ROUGE metric for measuring the performance of automatic summarization and machine translation tasks.

I hope you've enjoyed the article, let me know if you have any questions or suggestions via Twitter or in the comments below! If you're interested in more content like this I post on YouTube too.

Thanks for reading!

References

C. Lin, ROUGE: A Package for Automatic Evaluation of Summaries (2004), ACL

NLP With Transformers Course

**All images are by the author except where stated otherwise*

Enjoy the read? Reward the writer.[Beta]

Your tip will go to James Briggs through a third-party platform of their choice, letting them know you appreciate their story.

Give a tip

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Get this newsletter

About Help Terms Privacy