CharCut: Human-Targeted Character-Based MT Evaluation with Loose Differences

Adrien LARDILLEUX* and Yves LEPAGE**

*Fujitsu Technology Solutions / DGT, European Commission, Luxembourg
**IPS, Waseda University, Japan
adrien.lardilleux@ext.ec.europa.eu, yves.lepage@waseda.jp

早稲田大学 情報生産システム研究科
Graduate School of Information, Production and Systems, Waseda University

Background: metrics for MT

Trade-off between
- ease of use and
- correlation with human judgment

Background: metrics for MT

Trade-off between
ease of use and
correlation with human judgment

- Light methods
  (e.g., BLEU, WER)
  - Are very easy to use
    Are better fitted to languages with less resource

Background: metrics for MT

Trade-off between
ease of use and
correlation with human judgment

- Light methods
  (e.g., BLEU, WER)
  - Are very easy to use
    Are better fitted to languages with less resource
- Trained or knowledge-based metrics
  (e.g., BEER, DPMFcomb, UoW.ReVal)
  - Better correlate with human judgment
  - But need training or resources (e.g., paraphrase tables)

Background: metrics for MT

Trade-off between
- use of characters and
- use of words

Background: metrics for MT

Trade-off between
use of characters and
use of words

- Word-based methods
  (e.g., BLEU, WER)
  - Are well-fitted for languages like English or segmented Chinese

Background: metrics for MT

Trade-off between
use of characters and
use of words

- Word-based methods
  (e.g., BLEU, WER)
  - Are well-fitted for languages like English or segmented Chinese
- Character-based methods
  (e.g., CHRF, CHARACTER)
  - Are usually subject to noise for languages using the Latin script
  - But are better fitted for morphologically rich languages
    Better correlate with human judgments

Background: metrics for MT

Trade-off between
- ease of visualisation and
- the scoring mechanism

Background: metrics for MT

Trade-off between
ease of visualisation and
the scoring mechanism

- Word-based methods
  (e.g., TER, METEOR)
  - Allow to naturally derive user-friendly visual correspondences between candidate and reference translations

Background: metrics for MT

Trade-off between
ease of visualisation and
the scoring mechanism

- Word-based methods
  (e.g., TER, METEOR)
  - Allow to naturally derive user-friendly visual correspondences between candidate and reference translations
- Overlapping N-gram-based approaches
  (e.g., BLEU or CHRF)
  - Are more difficult to visualise

Proposal

CharCut, a light character-based machine translation evaluation metric derived from a human-targeted segment difference visualisation algorithm.

Proposal

CharCut, a light character-based machine translation evaluation metric derived from a human-targeted segment difference visualisation algorithm.

- Light automatic metric for MT output:
  no training, no use of extra knowledge

Proposal

CharCut, a light character-based machine translation evaluation metric derived from a human-targeted segment difference visualisation algorithm.

- Light automatic metric for MT output:
  no training, no use of extra knowledge
- High correlation with human judgment:
  on par with trained or knowledge-based metrics
  $\simeq$ best "untrained" metrics and $\gg$ BLEU and TER

Proposal

CharCut, a light character-based machine translation evaluation metric derived from a human-targeted segment difference visualisation algorithm.

- Light automatic metric for MT output:
  no training, no use of extra knowledge
- High correlation with human judgment:
  on par with trained or knowledge-based metrics
  $\simeq$ best "untrained" metrics and $\gg$ BLEU and TER
- Meaningful visualisation of MT output vs. human reference:
  scores directly reflect human-readable string differences

Method

Combination of

- iterative search for longest common substrings
  between candidate and reference translation
- simple length-based threshold
  $\Rightarrow$ loose differences $\Rightarrow$ less noisy character matches.

Method

Combination of

- iterative search for longest common substrings
  between candidate and reference translation
- simple length-based threshold
  ⇒ loose differences ⇒ less noisy character matches.

C: It␣was␣also␣remarkable␣for␣personal␣reasons.

R: It␣was␣noteworthy␣because␣of␣personal␣reasons.

Method

Combination of

- iterative search for longest common substrings
  between candidate and reference translation
- simple length-based threshold
  ⇒ loose differences ⇒ less noisy character matches.

C: It␣was␣also␣remarkable␣for␣personal␣reasons.

R: It␣was␣noteworthy␣because␣of␣personal␣reasons.

Method

Combination of

- iterative search for longest common substrings
  between candidate and reference translation
- simple length-based threshold
  $\Rightarrow$ loose differences $\Rightarrow$ less noisy character matches.

C: It␣was␣also␣remarkable␣for␣personal␣reasons.
R: It␣was␣noteworthy␣because␣of␣personal␣reasons.

Method and problem

Combination of

- iterative search for longest common substrings
  between candidate and reference translation
- simple length-based threshold
  ⇒ loose differences ⇒ less noisy character matches.

C: It␣was␣also␣remarkable␣for␣personal␣reasons.
R: It␣was␣noteworthy␣because␣of␣personal␣reasons.

Method and problem

Combination of

- iterative search for longest common substrings
  between candidate and reference translation
- simple length-based threshold
  ⇒ loose differences ⇒ less noisy character matches.

C: It␣was␣also␣remarkable␣for␣personal␣reasons.
R: It␣was␣noteworthy␣because␣of␣personal␣reasons.

Method and problem

Combination of

- iterative search for longest common substrings
  between candidate and reference translation
- simple length-based threshold
  ⇒ loose differences ⇒ less noisy character matches.

C: It␣was␣also␣remarkable␣for␣personal␣reasons.
R: It␣was␣noteworthy␣because␣of␣personal␣reasons.

Actual visualisation output: Russian–English

Seg. id	Score	Segment comparison: Deletion Insertion Shift
1	33/109= 30%	Src: *28-летний повар найден мертвым в торговом центре Сан-Франциско* MT: 28-year-old chef found dead in San Francisco shopping centre Ref: 28-Year-Old Chef Found Dead at San Francisco Mall
2	31/249= 12%	Src: *28-летний повар, который недавно переехал в Сан-Франциско, был найден мертвым в лестничном пролете местного торгового центра на этой неделе.* MT: the 28-year-old chef, who has recently moved to San Francisco, was found dead in the stairwell of a local shopping centre this week. Ref: A 28-year-old chef who had recently moved to San Francisco was found dead in the stairwell of a local mall this week.
3	111/262= 42%	Src: *Однако брат жертвы говорит, что он не может вообразить кого-то, кто желал бы причинить ему боль, отмечая: "Наконец-то дела у него шли на лад".* MT: However, the victim's brother says he can't imagine anyone who would wish to cause him pain, noting: "Finally he went on the lad." Ref: But the victim's brother says he can't think of anyone who would want to hurt him, saying, "Things were finally going well for him."

Actual visualisation output: English–German

6	150/489= **31%**	Src:	*The victim's brother, Louis Galicia, told ABC station KGO in San Francisco that Frank, previously a line cook in Boston, had landed his dream job as line chef at San Francisco's Sons & Daughters restaurant six months ago.*
		MT:	Der Bruder des Opfers, Louis Galicien, erzählte ABC-Station KGO in San Francisco, dass Frank, zuvor ein Line-Koch in Boston, seinen Traumjob als Linienchef im Restaurant Sons & Daughters von San Francisco vor sechs Monaten gelandet hatte.
		Ref:	Der Bruder des Opfers, Louis Galicia, teilte dem ABS Sender KGO in San Francisco mit, dass Frank, der früher als Koch in Boston gearbeitet hat, vor sechs Monaten seinen Traumjob als Koch im Sons & Daughters Restaurant in San Francisco ergattert hatte.
7	69/211= **33%**	Src:	*A spokesperson for Sons & Daughters said they were "shocked and devastated" by his death.*
		MT:	Eine Sprecherin von Sons & Daughters sagte, sie seien durch seinen Tod "geschockt und verwüstet" worden.
		Ref:	Ein Sprecher des Sons & Daughters sagte, dass sie über seinen Tod "schockiert und am Boden zerstört seien".

Method description

CharCut consists of three phases:

1. an iterative segmentation
   by longest common substrings
   between the candidate and the reference translations;
2. the identification of string shifts;
3. a scoring phase
   based on the lengths of remaining differences.

Recursive search

Recursive character-based longest-first approach, starting with
C_0 = the MT output segment and
R_0 = the human reference segment.

$$\begin{aligned} C_{n+1} &= C_n - \mathrm{LCSubstr}(C_n, R_n) \\ R_{n+1} &= R_n - \mathrm{LCSubstr}(C_n, R_n) \end{aligned} \tag{1}$$

Problem with character-based longest-first approach

Problem: Counter-intuitive segmentation.

C: [. . .] der␣Europäischen␣Gemeinsamen Strategie zur Unterstützung Palästinas [. . .]
R: [. . .] der␣Gemeinsamen␣Europäischen Strategie zur Unterstützung Palästinas [. . .]

Problem with character-based longest-first approach

Problem: Counter-intuitive segmentation.

C: [. . .] der␣Europäischen␣Gemeinsamen Strategie zur Unterstützung Palästinas [. . .]
R: [. . .] der␣Gemeinsamen␣Europäischen Strategie zur Unterstützung Palästinas [. . .]

- The same ending is shared by the two swapped words *Europäischen* and *Gemeinsamen*;
- This ending has been integrated into the LCSubstr;
- This prevents the more natural full word matches.

Problem with character-based longest-first approach

Problem: Counter-intuitive segmentation.

C: [. . .] der␣Europäischen␣Gemeinsamen Strategie zur Unterstützung Palästinas [. . .]
R: [. . .] der␣Gemeinsamen␣Europäischen Strategie zur Unterstützung Palästinas [. . .]

- The same ending is shared by the two swapped words *Europäischen* and *Gemeinsamen*;
- This ending has been integrated into the LCSubstr;
- This prevents the more natural full word matches.

Answer: Making the method aware of word separators.

Making the method aware of word separators

When searching for LCSubstr, consider only substr. of C_0 and R_0 of the three following types:

Making the method aware of word separators

When searching for LCSubstr, consider only substr. of C_0 and R_0 of the three following types:

- Substring inside one word only, including spaces and punctuations

Ex.: Hello,␣world!!!

Making the method aware of word separators

When searching for LCSubstr, consider only substr. of C_0 and R_0 of the three following types:

- Substring inside one word only, including spaces and punctuations

  Ex.: Hello,␣world!!!

- Several entire words, including beginning and end spaces or punctuations

  Ex.: Hello,␣world!!!

Making the method aware of word separators

When searching for LCSubstr, consider only substr. of C_0 and R_0 of the three following types:

- Substring inside one word only, including spaces and punctuations

  Ex.: Hello,␣world!!!

- Several entire words, including beginning and end spaces or punctuations

  Ex.: Hello,␣world!!!

- Run of non-word characters

  Ex.: Hello,␣world!!!

Longest common prefixes and suffixes

- The longest common prefix and the longest common suffix between C_0 and R_0 are added to the list of LCSubstr's, independently of their length
  - providing they match the second or third regular expression and
  - were not already extracted as a regular LCSubstr.
- This fixes frequent cases of true negatives
  - such as final punctuations or
  - segments shorter than the minimum match size which are usually felt as matches.

Longest common prefixes and suffixes

- The longest common prefix and the longest common suffix between C_0 and R_0 are added to the list of LCSubstr's, independently of their length
  - providing they match the second or third regular expression and
  - were not already extracted as a regular LCSubstr.
- This fixes frequent cases of true negatives
  - such as final punctuations or
  - segments shorter than the minimum match size which are usually felt as matches.
- Experiments showed no impact in terms of correlation with human judgement.

End of iterative segmentation

- Stop when length(LCSubstr(C_n, R_n)) $<$ some threshold
  (typically 3)
- Add longest common prefixes and suffixes.
- The set of LCSubstr's extracted up to last step n
  (including longest common prefix and suffix)
  are matches;
- The remaining strings,
  i.e., the last computed C_n and R_n,
  are loose differences.

Example of iterative search for longest common substrings

n	C_n	R_n	LCSubstr(C_n, R_n)	length
0	Before_the_game,_it_had_arrived_at_the_stadium_to_riots.	Before_the_match_there_was_a_riot_in_the_stadium.		

Example of iterative search for longest common substrings

n	C_n	R_n	LCSubstr(C_n, R_n)	length
0	Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.		
1	Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.	␣the␣stadium	12

Example of iterative search for longest common substrings

n	C_n	R_n	LCSubstr(C_n, R_n)	length
0	Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.		
1	Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.	`␣the␣stadium`	12
2	Before␣the␣game,␣it␣had␣arrived␣at |␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in|.	`Before␣the␣`	11

Example of iterative search for longest common substrings

n	C_n	R_n	LCSubstr(C_n, R_n)	length
0	Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.		
1	Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.	`␣the␣stadium`	12
2	Before␣the␣game,␣it␣had␣arrived␣at |␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in|.	`Before␣the␣`	11
3	|game,␣it␣had␣arrived␣at|␣to␣riots.	|match␣there␣was␣a␣riot␣in|.	`␣riot`	5

Example of iterative search for longest common substrings

<table>
<tr><th>n</th><th>C_n</th><th>R_n</th><th>LCSubstr(C_n, R_n)</th><th>length</th></tr>
<tr><td>0</td><td>Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.</td><td>Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.</td><td></td><td></td></tr>
<tr><td>1</td><td>Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.</td><td>Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.</td><td>␣the␣stadium</td><td>12</td></tr>
<tr><td>2</td><td>Before␣the␣game,␣it␣had␣arrived␣at |␣to␣riots.</td><td>Before␣the␣match␣there␣was␣a␣riot ␣in|.</td><td>Before␣the␣</td><td>11</td></tr>
<tr><td>3</td><td>|game,␣it␣had␣arrived␣at|␣to␣riots.</td><td>|match␣there␣was␣a␣riot␣in|.</td><td>␣riot</td><td>5</td></tr>
<tr><td>4</td><td>|game,␣it␣had␣arrived␣at|␣to|s.</td><td>|match␣there␣was␣a|␣in|.</td><td>at</td><td>2</td></tr>
</table>

Example of iterative search for longest common substrings

n	C_n	R_n	LCSubstr(C_n, R_n)	length
0	Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.		
1	Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.	␣the␣stadium	12
2	Before␣the␣game,␣it␣had␣arrived␣at |␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in|.	Before␣the␣	11
3	|game,␣it␣had␣arrived␣at|␣to␣riots.	|match␣there␣was␣a␣riot␣in|.	␣riot	5

Example of iterative search for longest common substrings

n	C_n	R_n	LCSubstr(C_n, R_n)	length
0	Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.		
1	Before␣the␣game,␣it␣had␣arrived␣at ␣the␣stadium␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in␣the␣stadium.	␣the␣stadium	12
2	Before␣the␣game,␣it␣had␣arrived␣at |␣to␣riots.	Before␣the␣match␣there␣was␣a␣riot ␣in|.	Before␣the␣	11
3	|game,␣it␣had␣arrived␣at|␣to␣riots.	|match␣there␣was␣a␣riot␣in|.	␣riot	5
4	|game,␣it␣had␣arrived␣at|␣to|s.	|match␣there␣was␣a|␣in|.	.	1

Example of segmentation

C_0: Before␣the␣game,␣it␣had␣arrived␣at␣the␣stadium␣to␣riots.

R_0: Before␣the␣match␣there␣was␣a␣riot␣in␣the␣stadium.

- LCSubstr's are in black.
- Remaining substrings (in red and blue) are loose differences.

Visualising string shifts

C_0: Before␣the␣game,␣it␣had␣arrived␣at␣the␣stadium␣to␣**riot**s.

R_0: Before␣the␣match␣there␣was␣a␣**riot**␣in␣the␣stadium.

- Here, `␣the␣stadium` and `␣riot` are crossed.
- For the purpose of visualisation,
  - the shortest one (`␣riot`) is marked as a shift,
  - and the other one as a regular match.

Identifying string shifts

$$C_{\text{match}} = \texttt{Before␣the␣}|\texttt{␣the␣stadium}|\texttt{␣riot}|\texttt{.}$$
$$R_{\text{match}} = \texttt{Before␣the␣}|\texttt{␣riot}|\texttt{␣the␣stadium}|\texttt{.}$$

To identify string shifts:

- determine longest subsequence of tokens (LCStr's)
- longest is defined in number of chars, not tokens.
  Here: `Before␣the␣|␣the␣stadium|.` (12+11+1=24 chars)

Identifying string shifts

$$C_{\text{match}} = \texttt{Before␣the␣|␣the␣stadium|␣riot|.}$$
$$R_{\text{match}} = \texttt{Before␣the␣|␣riot|␣the␣stadium|.}$$

To identify string shifts:

- determine longest subsequence of tokens (LCStr's)
- longest is defined in number of chars, not tokens.
  Here: `Before␣the␣|␣the␣stadium|.` (12+11+1=24 chars)

Regular matches / shifts:

- Tokens in longest subsequence are regular matches.
- Tokens outside of longest subsequence are **shifts**.
  Here: `␣riot`.

Scoring scheme

Result of the iterative segmentation and identification of shifts: segmentation of input segments in 3 types of substrings:

- regular matches
- **shifts**
- loose differences, i.e.,
  - deletions from the candidate segment
  - insertions into the reference segment

Scoring scheme

Result of the iterative segmentation and identification of shifts: segmentation of input segments in 3 types of substrings:

$$\text{Score} \propto \#\text{deletions} + \#\text{insertions} + \#\textbf{shifts}$$

- regular matches
- **shifts**
- loose differences, i.e.,
  - deletions from the candidate segment
  - insertions into the reference segment

Scoring scheme

Result of the iterative segmentation and identification of shifts: segmentation of input segments in 3 types of substrings:

$$\text{Score} \propto \#\text{deletions} + \#\text{insertions} + \#\textbf{shifts}$$

	Indiv. score
▶ regular matches	0
▶ **shifts**	
▶ loose differences, i.e.,	

- deletions from the candidate segment
- insertions into the reference segment

Scoring scheme

Result of the iterative segmentation and identification of shifts: segmentation of input segments in 3 types of substrings:

$$\text{Score} \propto \#\text{deletions} + \#\text{insertions} + \#\textbf{shifts}$$

	Indiv. score
▸ regular matches	0
▸ **shifts**	
▸ loose differences, i.e.,	
▸ deletions from the candidate segment	1
▸ insertions into the reference segment	1

Scoring scheme

Result of the iterative segmentation and identification of shifts: segmentation of input segments in 3 types of substrings:

$$\text{Score} \propto \#\text{deletions} + \#\text{insertions} + \#\textbf{shifts}$$

	Indiv. score
▸ regular matches	0
▸ **shifts** (counted once although appear in both segments)	1
▸ loose differences, i.e.,	
▸ deletions from the candidate segment	1
▸ insertions into the reference segment	1

Optimizing for correlation with human judgement

Two different normalisations:

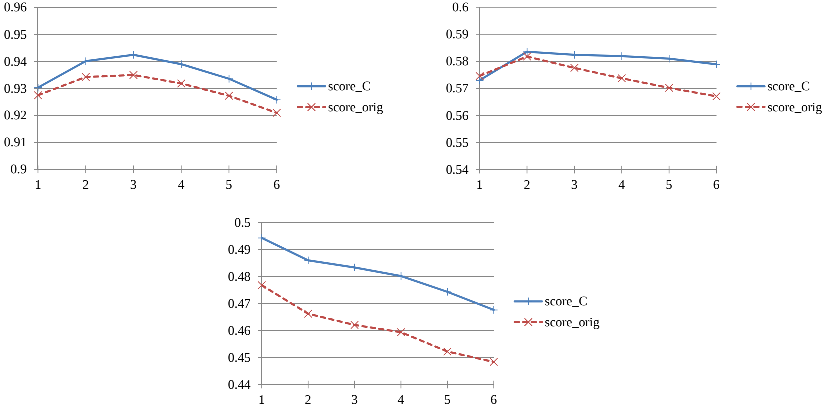
- total length of candidate and reference (intuitive)
  $\Rightarrow$ score between 0 and 1:

$$\text{score}_{\text{orig}} = \frac{\#\text{deletions} + \#\text{insertions} + \#\text{shifts}}{|C_0| + |R_0|} \quad (2)$$

- length of candidate only (Wang et al., 2016)
  $\Rightarrow$ higher correlation with human judgements

$$\text{score}_{C} = \min\left(1, \frac{\#\text{deletions} + \#\text{insertions} + \#\text{shifts}}{2 \times |C_0|}\right) \quad (3)$$

Pearson correlation for the two scoring schemes



Absolute Pearson correlation against minimum match size in characters (length-based threshold) (system DA, segment-DA, segment-HUME)

Comparison

- With metrics that took part in WMT16 tasks
  - system-level DA
  - segment-level DA
  - segment-level HUME
- Criterion: average Pearson correlation coefficients over all language pairs.

Comparison

- With metrics that took part in WMT16 tasks
  - system-level DA
  - segment-level DA
  - segment-level HUME
- Criterion: average Pearson correlation coefficients over all language pairs.

Notations:

- Brackets = metrics that did not participate in the English-to-Russian evaluation (i.e., one less figure used);
- Asterisks = our own runs;
- Everything else = figures from (Bojar et al., 2016).

System-level DA

Metric	Avg. corr. ± stddev.
UoW.ReVaL	(0.972 ± 0.013)
MPEDA	0.945 ± 0.044
*CharCut	0.942 ± 0.037
chrF2	0.934 ± 0.038
chrF3	0.934 ± 0.035
*Lev. distance	0.930 ± 0.049
BEER	0.928 ± 0.054
chrF1	0.927 ± 0.051
CharacTER	0.922 ± 0.055
mtevalNIST	0.886 ± 0.068
mtevalBLEU	0.867 ± 0.060
⋮	
mosesCDER	0.861 ± 0.061
mosesTER	0.851 ± 0.061
mosesPER	0.842 ± 0.096
wordF3	0.836 ± 0.069
wordF2	0.836 ± 0.069
wordF1	0.831 ± 0.071
mosesWER	0.812 ± 0.099
mosesBLEU	0.810 ± 0.082

Segment-level DA

Metric	Avg. corr. ± stddev.
DPMFCOMB	(0.633 ± 0.048)
METRICS-F	(0.631 ± 0.049)
COBALT-F.	(0.617 ± 0.040)
MPEDA	0.584 ± 0.053
*CHARCUT	0.582 ± 0.076
UPF-COBALT	(0.582 ± 0.060)
CHRF3	0.560 ± 0.082
CHRF2	0.559 ± 0.081
*Lev. distance	0.556 ± 0.065
BEER	0.556 ± 0.082
CHRF1	0.548 ± 0.079
*CharacTER	0.537 ± 0.074
UOW.REVAL	0.530 ± 0.035
⋮	
WORDF3	0.524 ± 0.055
WORDF2	0.522 ± 0.055
WORDF1	0.514 ± 0.055
SENTBLEU	0.510 ± 0.039
*TER	0.485 ± 0.052
DTED	0.330 ± 0.058

Segment-level HUME

Metric	Avg. corr. ± stddev.
chrF3	0.519 ± 0.096
chrF2	0.517 ± 0.092
BEER	0.513 ± 0.079
chrF1	0.503 ± 0.079
MPEDA	0.492 ± 0.073
*CharCut	0.483 ± 0.121
wordF3	0.452 ± 0.092
wordF2	0.450 ± 0.091
wordF1	0.439 ± 0.088
*CharacTER	0.438 ± 0.126
*Lev. distance	0.437 ± 0.109
sentBLEU	0.401 ± 0.101
*TER	0.394 ± 0.125

Analysis of the comparison with other metrics

- High correlation with human judgment

Analysis of the comparison with other metrics

- High correlation with human judgment
- Comparison with light metrics:
  - Top average correl. on system- and segment-level DA eval. compared with CHRF, WORDF, CharacTER
  - Much higher correl. than BLEU and TER

Analysis of the comparison with other metrics

- High correlation with human judgment
- Comparison with light metrics:
  - Top average correl. on system- and segment-level DA eval. compared with chrF, wordF, CharacTER
  - Much higher correl. than BLEU and TER
- Comparison with trained metrics:
  - On par with MPEDA (relies on additional training corpora)

Speed

On a 2.8 GHz processor, for Python implementations:

Metric	segments/s
chrF	600
CharCut	260
CharacTER	54

Conclusion

CharCut: character-based machine translation evaluation metric.

- Is language independent.
- Requires no additional resource or training.

Conclusion

CharCut: character-based machine translation evaluation metric.

- Is language independent.
- Requires no additional resource or training.
- Relies on loose differences,
  residuals of iterative search for longest common substrings.
- Was initially designed for displaying differences
  between reference and candidate segments to end users.
- Produces scores that directly reflect differences.

Conclusion

CharCut: character-based machine translation evaluation metric.

- Is language independent.
- Requires no additional resource or training.
- Relies on loose differences,
  residuals of iterative search for longest common substrings.
- Was initially designed for displaying differences
  between reference and candidate segments to end users.
- Produces scores that directly reflect differences.
- Exhibits good correlation with human judgement.

Conclusion

CharCut: character-based machine translation evaluation metric.

- Is language independent.
- Requires no additional resource or training.
- Relies on loose differences,
  residuals of iterative search for longest common substrings.
- Was initially designed for displaying differences
  between reference and candidate segments to end users.
- Produces scores that directly reflect differences.
- Exhibits good correlation with human judgement.

Good visual representation $\Rightarrow$ High correlation with human judgement

Future work

- Finer handling of shifts
  as CharCut is currently unaware of shift distance;
- Automatic correlation of the minimum match size
  with the number of highlighted substrings
  in order to keep outputs readable even with very different input segments.

Availability

CharCut is open source and available at

https://github.com/alardill/CharCut.

It consists of a single Python script that computes scores and highlights differences (HTML outputs).

Seg. id	Score	Segment comparison: Deletion Insertion Shift
1	19/50= 38%	MT: Thank you for listening. Ref: Thanks for your attention. match: 5
Total	19/50= 38%	

Interface convention

- The interface is kept slick on purpose.
- It uses only classical colours:
  - red for deletions,
  - blue for insertions,
  - bold for **shifts**,
  - yellow background for matching substrings when pointed with the mouse.
- The scores directly reflect the number of highlighted characters.

HTML sample output (WMT17 English-Chinese, 2-char min match size)

Seg. id	Score	Segment comparison: **Deletion** **Insertion** **Shift**
1	27/39= 69%	Src: *28-Year-Old Chef Found Dead at San Francisco Mall* MT: 28岁的 Chef Fand 死在旧金山商城 Ref: 28岁厨师被发现死于旧金山一家商场
2	21/69= 30%	Src: *A 28-year-old chef who had recently moved to San Francisco was found dead in the stairwell of a local mall this week.* MT: 一名最近搬到旧金山的28岁厨师，本周在当地一家商场的楼梯间被发现死亡。 Ref: 近日刚搬至旧金山的一位28岁厨师本周被发现死于当地一家商场的楼梯间。 match: 5
3	44/72= 61%	Src: *But the victim's brother says he can't think of anyone who would want to hurt him, saying, "Things were finally going well for him."* MT: 但受害人的哥哥说，他不能想到任何人都想伤害他，说：“事情终于对他有利了。” Ref: 但受害人的哥哥表示想不出有谁会想要加害于他，并称“一切终于好起来了。”
Total	92/180= 51%	

References

Bojar, O., Graham, Y., Kamran, A., and Stanojević, M. (2016). Results of the WMT16 Metrics Shared Task. In *Proceedings of the First Conference on Machine Translation*, pages 199–231, Berlin, Germany. Association for Computational Linguistics.

Wang, W., Peter, J.-T., Rosendahl, H., and Ney, H. (2016). CharacTer: Translation Edit Rate on Character Level. In *Proceedings of the First Conference on Machine Translation*, pages 505–510, Berlin, Germany. Association for Computational Linguistics.