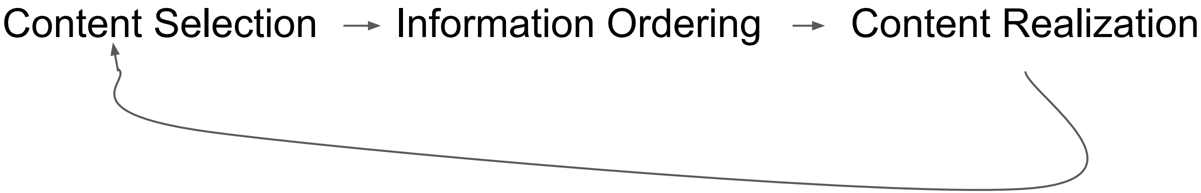
IMPROVED SYSTEM WITH INFO ORDERING

Long Cheng, Sheng Bi, Catherine Wang, Vicky Xiang, Carrie Yuan
University of Washington

System architecture



Content Selection → Information Ordering → Content Realization

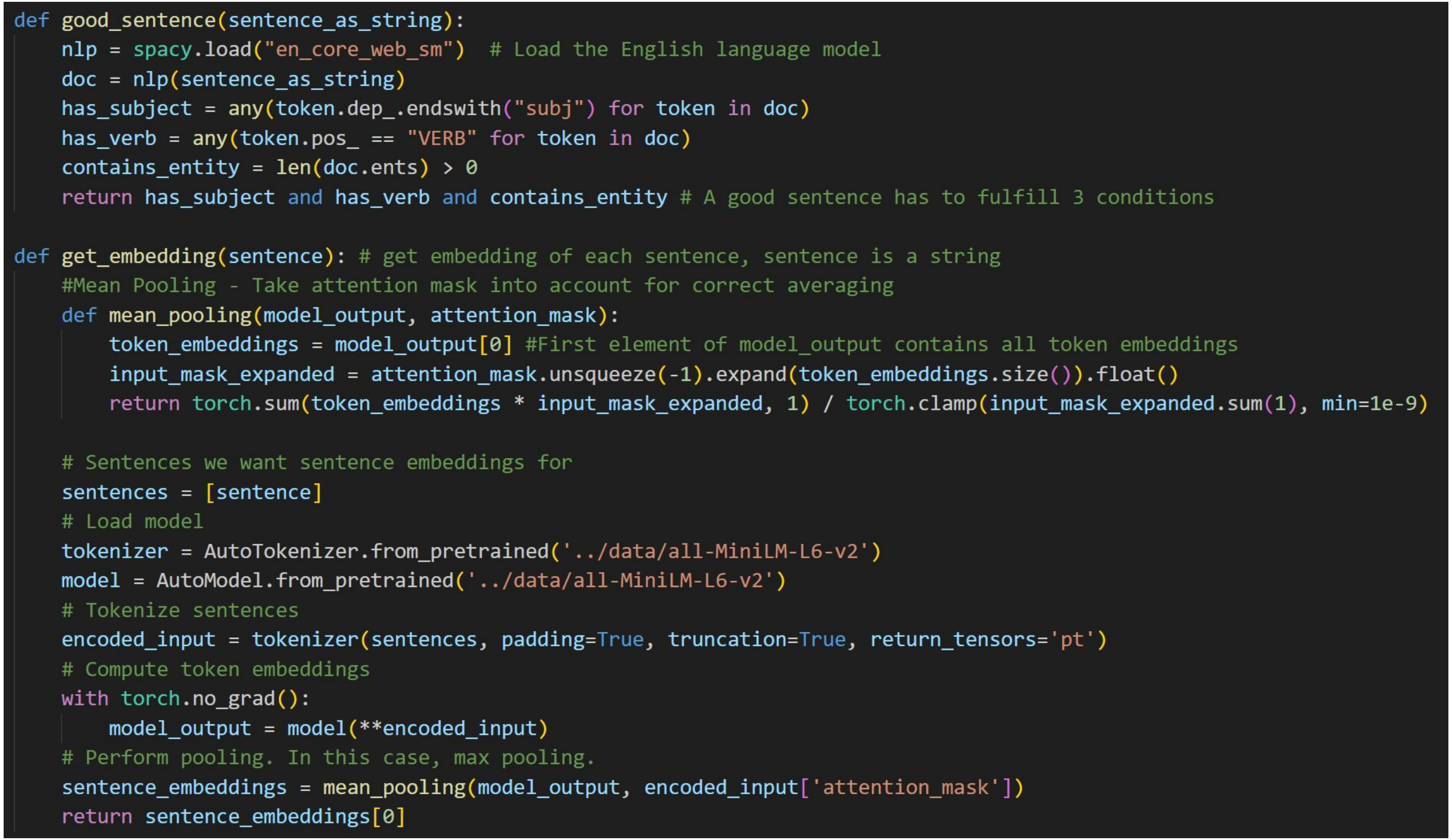
Approaches

Content Selection Improvement

&

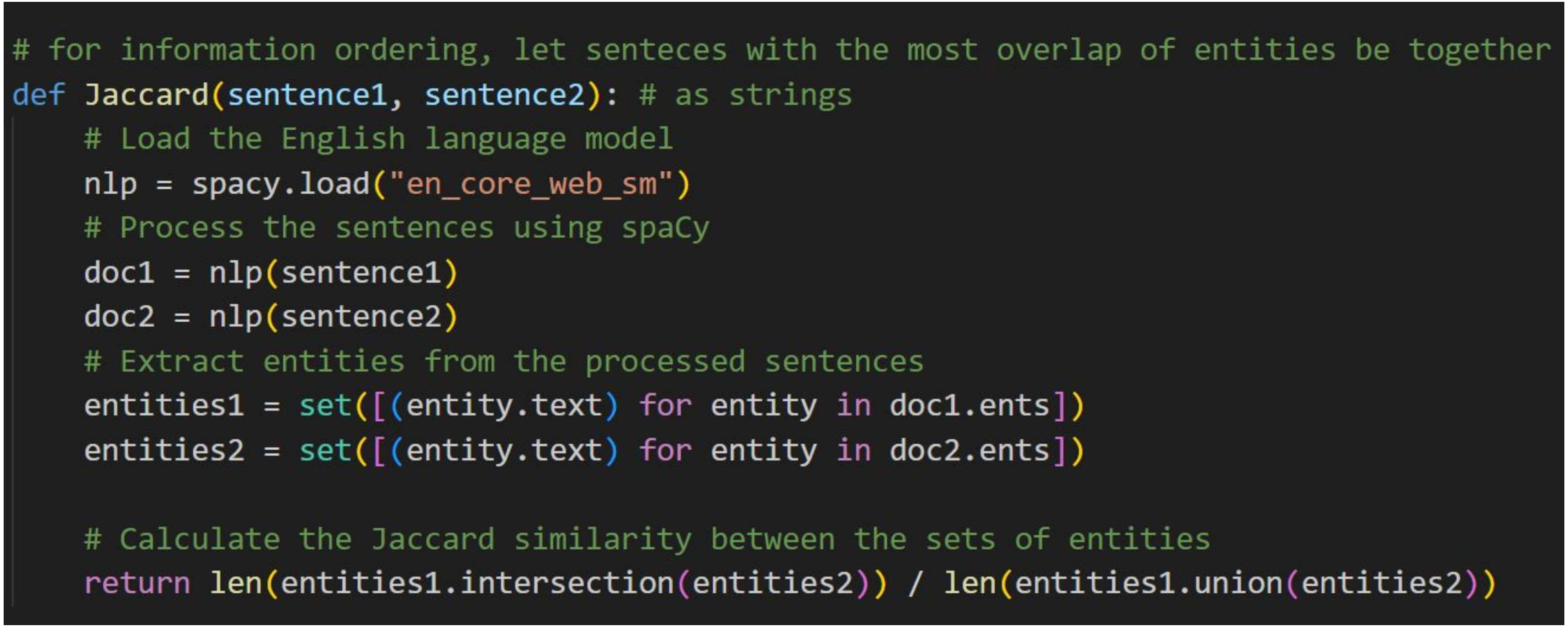
Information Ordering

LLR (Content Selection Improvement)



```python
def good_sentence(sentence_as_string):
    nlp = spacy.load("en_core_web_sm")  # Load the English language model
    doc = nlp(sentence_as_string)
    has_subject = any(token.dep_.endswith("subj") for token in doc)
    has_verb = any(token.pos_ == "VERB" for token in doc)
    contains_entity = len(doc.ents) > 0
    return has_subject and has_verb and contains_entity # A good sentence has to fulfill 3 conditions

def get_embedding(sentence): # get embedding of each sentence, sentence is a string
    #Mean Pooling - Take attention mask into account for correct averaging
    def mean_pooling(model_output, attention_mask):
        token_embeddings = model_output[0] #First element of model_output contains all token embeddings
        input_mask_expanded = attention_mask.unsqueeze(-1).expand(token_embeddings.size()).float()
        return torch.sum(token_embeddings * input_mask_expanded, 1) / torch.clamp(input_mask_expanded.sum(1), min=1e-9)

    # Sentences we want sentence embeddings for
    sentences = [sentence]
    # Load model
    tokenizer = AutoTokenizer.from_pretrained('../data/all-MiniLM-L6-v2')
    model = AutoModel.from_pretrained('../data/all-MiniLM-L6-v2')
    # Tokenize sentences
    encoded_input = tokenizer(sentences, padding=True, truncation=True, return_tensors='pt')
    # Compute token embeddings
    with torch.no_grad():
        model_output = model(**encoded_input)
    # Perform pooling. In this case, max pooling.
    sentence_embeddings = mean_pooling(model_output, encoded_input['attention_mask'])
    return sentence_embeddings[0]
```

LLR (Information Ordering - Jaccard similarity)



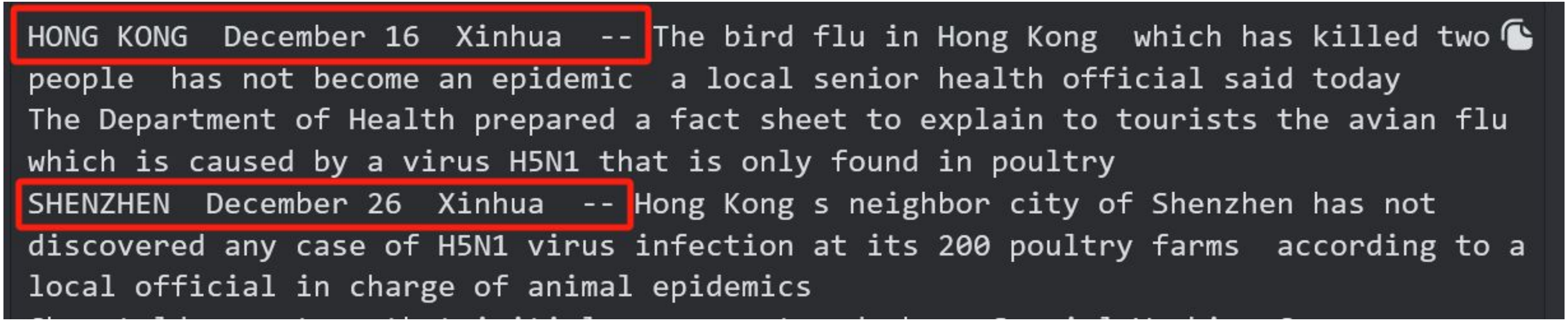
```python
# for information ordering, let senteces with the most overlap of entities be together
def Jaccard(sentence1, sentence2): # as strings
    # Load the English language model
    nlp = spacy.load("en_core_web_sm")
    # Process the sentences using spaCy
    doc1 = nlp(sentence1)
    doc2 = nlp(sentence2)
    # Extract entities from the processed sentences
    entities1 = set([(entity.text) for entity in doc1.ents])
    entities2 = set([(entity.text) for entity in doc2.ents])

    # Calculate the Jaccard similarity between the sets of entities
    return len(entities1.intersection(entities2)) / len(entities1.union(entities2))
```

Observations of LLR

When compared to model summaries typically comprising 5-7 sentences, the output summaries generated by LLR generally consist of 3-4 sentences, thereby reducing the significance of information ordering.

Interestingly, despite normalizing weights during the calculation of each sentence's weight, the algorithm demonstrates a preference for longer sentences over shorter ones.

An issue across all 3 methods



```
HONG KONG  December 16  Xinhua  -- The bird flu in Hong Kong  which has killed two
people  has not become an epidemic  a local senior health official said today
The Department of Health prepared a fact sheet to explain to tourists the avian flu
which is caused by a virus H5N1 that is only found in poultry
SHENZHEN  December 26  Xinhua  -- Hong Kong s neighbor city of Shenzhen has not
discovered any case of H5N1 virus infection at its 200 poultry farms  according to a
local official in charge of animal epidemics
```

SumBasic+

The following conditions are applied for content selections, based on the result from probability weights:

- Complete sentences: has a subject, has a verb
- Contains a named entity
- enlarged the set of stopwords (such as, say/said/told)
- removed duplicates (identical sentences from multiple files)

Information Ordering 1 - Entity Grid

Part 1: Preprocessing

- Entity selections
- Build entity grid for each file for any docSetA
- Turn entity grid (of each file) into a feature vector

Our Features:
✅used spacy to retrieve entities
✅used tfidf to shrink the size of entities (otherwise the probability score for each bigram entity would be too low)
❌have not managed to resolve coreference issues

Up to now, we have constructed our positive data.

Information Ordering 1 - Entity Grid

Part 2: Construct negative data

- reverse the order of the sentences of each file
- render the reordered file into an entity grid and build corresponding feature vector for this false rendering.

Our Features:

✅We only constructed one negative instance for each file: by reversing the original order - list_of_sentences[::-1]
✅To avoid analyzing all permutations of the file, we could, only reverse sentences which have high correlations.

Information Ordering 1 - Entity Grid

Part 3: Build feature vectors

Having constructed our positive data and negative data, we can build our training data matrix.

💣💣Be aware of the issue of collinearity. The dimension for bigram-entity probability vector is 15.

If we distinguish between salience classes (k), the dimension would be 15 * k

Information Ordering 1 - Entity Grid

Part 4: Run a default LinearSVC using scikit-learn to train the model.

Although the original paper only mentioned SVC, we have been thinking the possibility of testing that on an HMM

Information Ordering 1 - Entity Grid

Part 5: Build test data for prediction.

Step 1

- given the results from, e.g. sumbasic, we find all permutations of the sentences of each summary.

Step 2:

- render each reordered summary into a bigram-entity probability vector

Step 3:

- Predict using the trained model from the previous part, and use the result of SVC.decision_function(test_x[docSetA]) to select the summary with the largest score.

LexRank Improvement

The D3 results shows a few areas for improvement.

- The selected sentences tend to be short sentences.
  - Increase similarity threshold so that it is not over-voted on short sentences.
  - Selecting only sentences that are longer than 10 words.
  - Check for complete sentences with a subject and a verb

- Similarity matrix doesn’t account for synonym and paraphrasing.
  - Using pretrained transformer model to encode the sentence, so that the similarity score is a more accurate representation of semantic similarity between two sentences.

Information Ordering 2 - Chronological Ordering

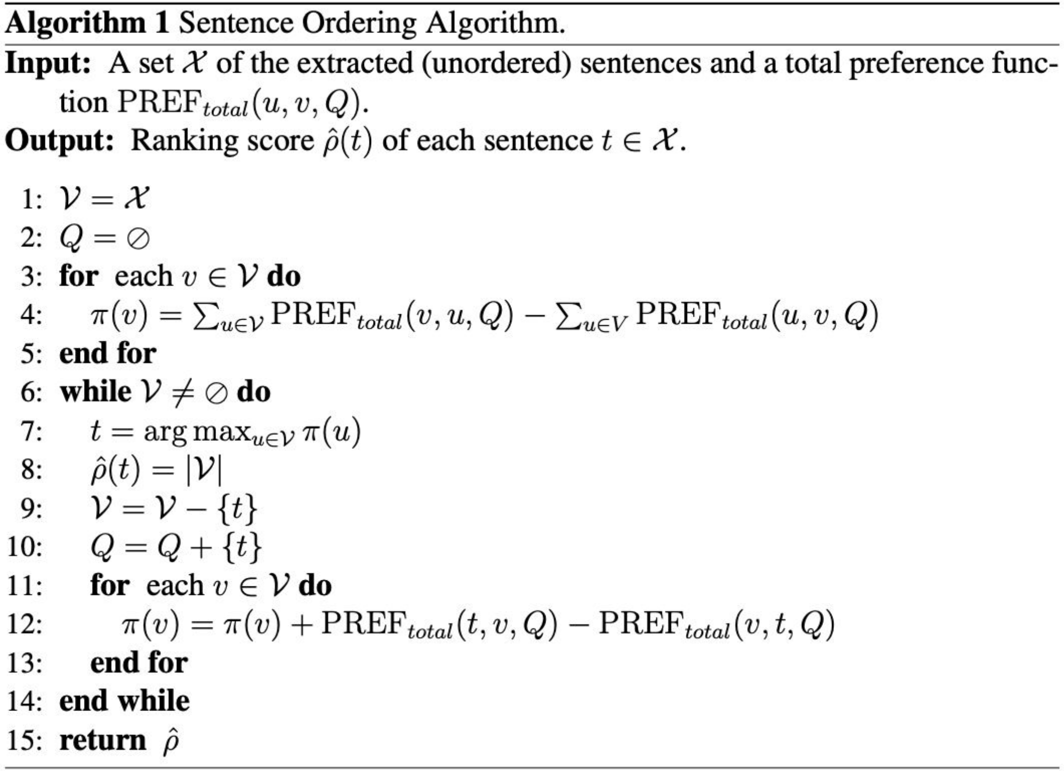
Step 1

- Preprocessing the sentences, by attaching the extracted datetime and sentence sequence number to the sentence object.

Step 2:

- Render the set of unordered sentences using the greedy ordering algorithm, using the simple datetime + sequence number comparison as preference function.

Information Ordering 2 - Chronological Ordering



Algorithm 1 Sentence Ordering Algorithm.

Input: A set $\mathcal{X}$ of the extracted (unordered) sentences and a total preference function $\mathrm{PREF}_{total}(u, v, Q)$.

Output: Ranking score $\hat{\rho}(t)$ of each sentence $t \in \mathcal{X}$.

```
$\mathcal{V} = \mathcal{X}$
$Q = \oslash$
for each $v \in \mathcal{V}$ do
    $\pi(v) = \sum_{u \in \mathcal{V}} \mathrm{PREF}_{total}(v, u, Q) - \sum_{u \in V} \mathrm{PREF}_{total}(u, v, Q)$
end for
while $\mathcal{V} \neq \oslash$ do
    $t = \arg\max_{u \in \mathcal{V}} \pi(u)$
    $\hat{\rho}(t) = |\mathcal{V}|$
    $\mathcal{V} = \mathcal{V} - \{t\}$
    $Q = Q + \{t\}$
    for each $v \in \mathcal{V}$ do
        $\pi(v) = \pi(v) + \mathrm{PREF}_{total}(t, v, Q) - \mathrm{PREF}_{total}(v, t, Q)$
    end for
end while
return $\hat{\rho}$
```

Evaluation

- ROUGE score
- InfoLM
- Automated pyramid evaluation
- Human evaluation

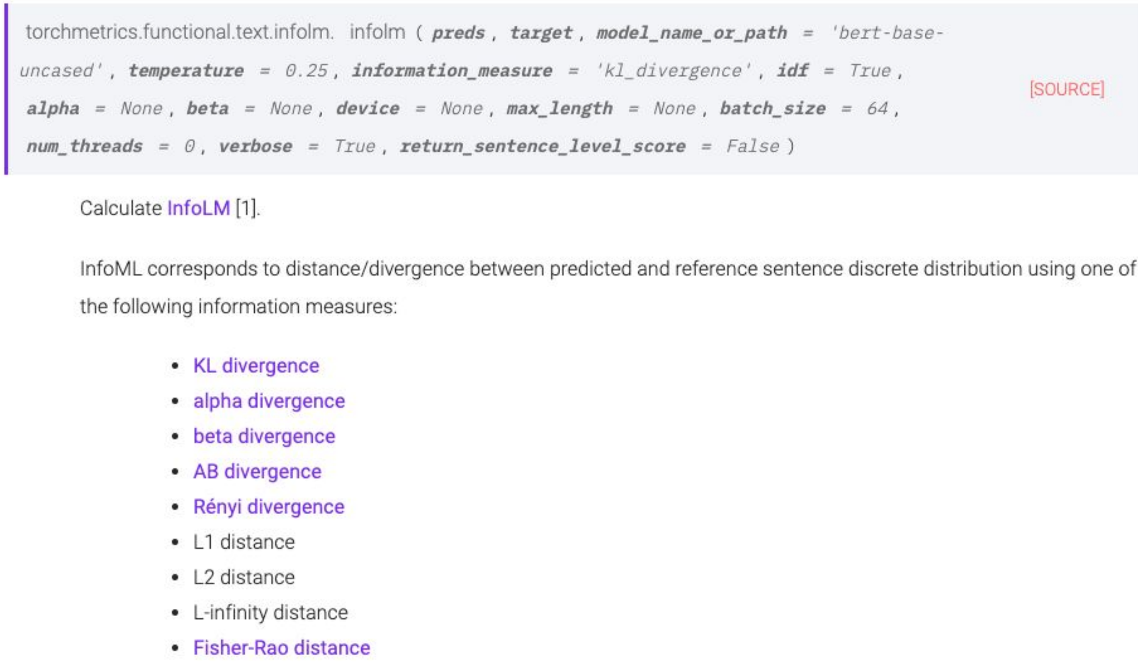
Results

Improved system	**ROUGE-1 R**	**% Change**	**ROUGE-2 R**	**% Change**
SumBasic	.26561	-23.8	.03623	-16.6
LLR	.24779	28.2	.03813	37.0
LexRank	.39722	26.7	.07308	22.4

Results

Improved system	ROUGE-1 Precision	% Change	ROUGE-2 Precision	% Change
SumBasic	.25970	45.8	.03552	60.2
LLR	.29220	19.2	.04460	27.7
LexRank	.24350	-12.4	.04414	-11.8

InfoLM Metrics



torchmetrics.functional.text.infolm. infolm (***preds*** , ***target*** , ***model_name_or_path*** = *'bert-base-uncased'* , ***temperature*** = *0.25* , ***information_measure*** = *'kl_divergence'* , ***idf*** = *True* , ***alpha*** = *None* , ***beta*** = *None* , ***device*** = *None* , ***max_length*** = *None* , ***batch_size*** = *64* , ***num_threads*** = *0* , ***verbose*** = *True* , ***return_sentence_level_score*** = *False*) [SOURCE]

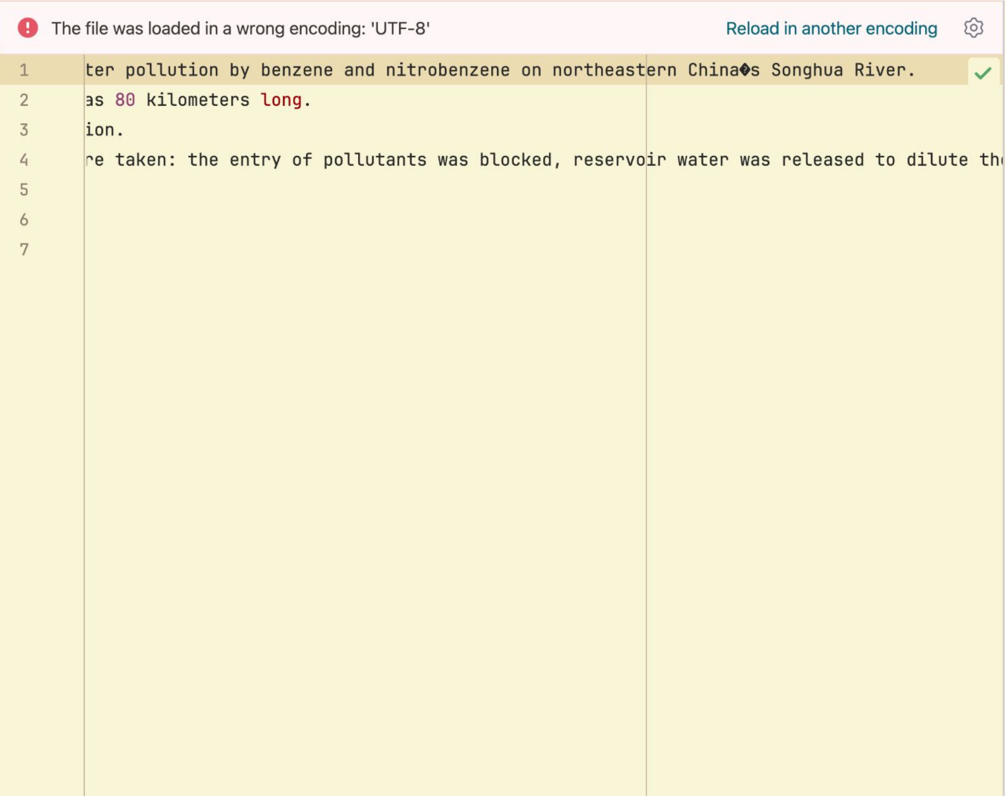
Calculate InfoLM [1].

InfoML corresponds to distance/divergence between predicted and reference sentence discrete distribution using one of the following information measures:

- KL divergence
- alpha divergence
- beta divergence
- AB divergence
- Rényi divergence
- L1 distance
- L2 distance
- L-infinity distance
- Fisher-Rao distance

https://lightning.ai/docs/torchmetrics/stable/text/infolm.html

Handling Encoding



The file was loaded in a wrong encoding: 'UTF-8'

```
ter pollution by benzene and nitrobenzene on northeastern China�s Songhua River.
as 80 kilometers long.
ion.
re taken: the entry of pollutants was blocked, reservoir water was released to dilute th



```

InfoLM Scores

Content Selection	KL Divergence (excluding outlier(s))	L1	L2	L_inf
LLR	-5.276104444	0.4212043478	0.07614347826	0.03005869565
SumBasic	-5.852161364	0.43845	0.07721304348	0.0295891304
LexRank	-4.740382609	0.4140065217	0.0744673913	0.02884130435