Supplementary Materials for Learning to Recognize Discontiguous Entities

Aldrian Obaja Muis   Wei Lu
Singapore University of Technology and Design
{aldrian_muis,luwei}@sutd.edu.sg

Abstract

This is the supplementary material for "Learning to Recognize Discontiguous Entities" [Muis and Lu, 2016]. This material gives more details in the experiment setup, the ambiguity of each model, and compare the models from theoretical point of view.

1 Model Ambiguity

This work attempts to define a more formal way to compare two models in terms of its ambiguity.

To compare the ambiguity between models, we first define a notion of *model ambiguity level*, which is defined as the average number of distinct interpretations across its set of *canonical encodings*. A canonical encoding is a fixed, selected representation of a particular set of mentions in a sentence, among (possibly) several alternative representations. Several alternatives may be present due to the ambiguity of the encoding-decoding process and also since the output of the model is not restricted to a specific rule. For example, for the text "John Smith", a model trained in BIO format might output "B-PER I-PER" or "I-PER I-PER", and both will still mean that "John Smith" is a person, although the "correct" encoding would of course be "B-PER I-PER", which is selected as the canonical encoding. Intuitively, a canonical encoding is a formal way to say that we only consider the "correct" encodings.

Note that by definition, the number of canonical encodings cannot be larger than the number of possible interpretations, so ambiguity level will be at least 1. When the number of canonical encodings is strictly smaller than the number of possible entity combinations, we have either ambiguity (when several entity combinations have the same canonical encoding) or incompleteness (when a entity combination does not have an encoding). A model which is complete and not ambiguous will have the lowest ambiguity level, which is 1. We note that in our case, all models are complete as for any entity combination each model is able to encode it. All models that we consider, however, are ambiguous since some in each model there are encodings that can be interpreted in more than one way.

Let $\mathcal{N}(n)$ be the number of all possible entity combinations in a sentence with n words, and $\mathcal{M}(n)$ be the number of canonical encodings in the model for a sentence with n words. Then the ambiguity level of a model is:

$$A = \lim_{n\to\infty} \frac{\sum_{i=1}^{n} \mathcal{N}(i)}{\sum_{i=1}^{n} \mathcal{M}(i)} \tag{1}$$

assuming the limit exists. If it does not exist, the ambiguity is undefined. This can mean two models which are incomparable, or that the number of possible entity combinations is much larger than the number of canonical encodings. When it exists, the values of A lie in the range $[1, \infty)$. Sometimes when $\mathcal{M}(n)$ is very small compared to $\mathcal{N}(n)$, it might be useful to compute the *log-ambiguity* instead, defined as:

$$\mathcal{A} = \lim_{n\to\infty} \frac{\log\left(\sum_{i=1}^{n} \mathcal{N}(i)\right)}{\log\left(\sum_{i=1}^{n} \mathcal{M}(i)\right)} \tag{2}$$

This is what we will use in the remainder of this article.

Relative Ambiguity For some tasks, the number of possible encodings in a practical model may be very small compared to the number of possible entity combinations resulting in very high ambiguity, and so it is difficult to compare two models. To overcome this, we define a notion of *relative ambiguity*, which is defined as the ratio of the ambiguity level of two models:

$$\mathcal{A}_r(M_1, M_2) = \frac{\mathcal{A}_{M_1}(n)}{\mathcal{A}_{M_2}(n)} = \lim_{n\to\infty} \frac{\log\sum_{i=1}^{n} \mathcal{M}_{M_2}(i)}{\log\sum_{i=1}^{n} \mathcal{M}_{M_1}(i)} \tag{3}$$

A relative ambiguity greater than one means the first model is more ambiguous compared to the second model. A relative ambiguity of 1 means the two models have the same level of ambiguity.

Now we calculate the number of canonical encodings $\mathcal{M}_{\text{LI}}(n)$, $\mathcal{M}_{\text{SH}}(n)$, and $\mathcal{M}_{\text{SP}}(n)$ for linear-chain, SHARED, and SPLIT model, respectively.

Calculating $\mathcal{M}_{\text{LI}}(n)$ For the linear-chain, the number of possible encodings is the number of possible tag sequence, which is 7^n in this case, since there are 7 possible tags. Note that this number is larger than the number of canonical encodings, since some of the tag sequences have the same interpretation and some are not valid. So: $\sum_{i=1}^{n} \mathcal{M}_{\text{LI}}(i) < \sum_{i=1}^{n} 7^i = \frac{7^{n+1}-7}{6} < 8^n < 2^{3n}$ and so $\log\sum_{i=1}^{n} \mathcal{M}_{\text{LI}}(i) < 3n\log 2$.

Calculating $\mathcal{M}_{\text{SH}}(n)$ and $\mathcal{M}_{\text{SP}}(n)$ For our models, the number of canonical encodings is equal to the number of valid subgraph in the model, since each distinct subgraph will yield distinct interpretation. This is quite tricky to calculate; a straightforward application of the standard dynamic programming algorithm similar to the inside algorithm that uses the nodes as states fails in this case because it also includes certain combinations not representable in our graph models. To calculate the number of distinct subgraph, we need to use a combination of nodes as states.

To explain this idea, we will calculate the number of subgraph of a simple graph shown in Figure 1.The nodes are indexed by position from right to left, starting from 1, and also by level from bottom to top, starting from 0, as in the figure. So the top left node has level 2 and position 6, and the bottom right node has level 0 and position 1. Let n_i^j denote the node at level i and position j. From each node n_i^j except the nodes in the bottom row and the rightmost in each level, there are two edges, one connecting to n_i^{j-1} and another connecting to n_{i-1}^{j-1}. Now we want to count the number of distinct connected directed acyclic graphs (DAGs) that include the top left node and the bottom right node.

First note that at each position, any combination of nodes in the three levels can be assigned a unique number from 0 to 7 by considering the chosen nodes as binary number, with the node at level 2 being the most significant bit. For example, the number 5, or $\overline{101}$ in binary, represents the set of two nodes at level 2 and level 0.

Now, we define 8 functions, $f_{\overline{000}}(n)$, $f_{\overline{001}}(n)$, $\ldots$, $f_{\overline{111}}(n)$, one for each combination of nodes at each position. The function $f_k(n)$ represents the number of directed acyclic graphs with the node combinations represented by k at position n as the sources, and the bottom right node as the sink. Then we have each

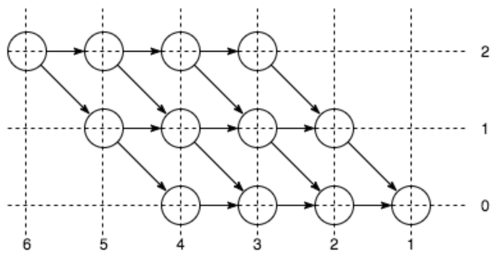

Figure 1: Simple graph

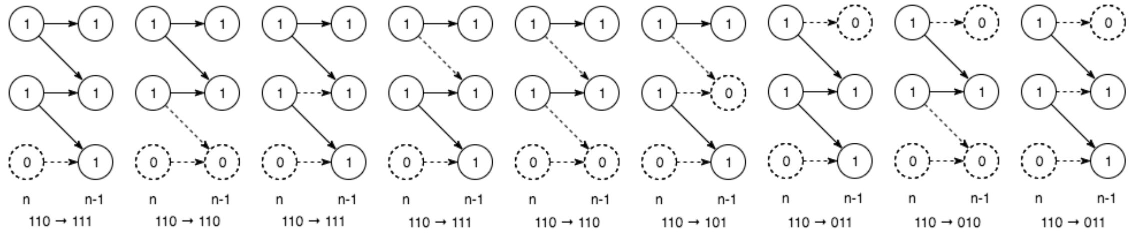

Figure 2: Reachable node states from the state $\overline{110}$

function as the sum over all reachable node combinations at the previous position. To find the reachable states, we enumerate all possible edge combinations for a given nodes configuration. Figure 2 shows the 9 reachable states from the state $\overline{110}$. Note that a node combination may be reachable in multiple ways.

From there, we have $f_{\overline{110}}(n) = f_{\overline{010}}(n-1) + 2f_{\overline{011}}(n-1) + f_{\overline{101}}(n-1) + 2f_{\overline{110}}(n-1) + 3f_{\overline{111}}(n-1)$. By representing this as transition vector, we have:

$$f_{\overline{110}}(n) = \begin{bmatrix} 0 & 0 & 1 & 2 & 0 & 1 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} f_{\overline{000}}(n-1) \\ f_{\overline{001}}(n-1) \\ f_{\overline{010}}(n-1) \\ f_{\overline{011}}(n-1) \\ f_{\overline{100}}(n-1) \\ f_{\overline{101}}(n-1) \\ f_{\overline{110}}(n-1) \\ f_{\overline{111}}(n-1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 2 & 0 & 1 & 2 & 3 \end{bmatrix} \times \mathbf{f}(n-1)$$

Stacking the transition vectors for all 8 functions, we have the following transition matrix $\mathbf{T}$:

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 3 & 0 & 1 & 0 & 5 \end{bmatrix} \text{, and} \qquad \mathbf{f}(n) = \mathbf{T} \cdot \mathbf{f}(n-1)$$

Using this recursive formulation, we can calculate the number of possible DAGs at the top left node by calculating $f_{\overline{100}}(n)$. Initially we have $f_{\overline{001}}(1) = 1$ and $f_k(1) = 0$ for $k \neq 1$. Then we have the following formulation for $f(n)$, the number of DAGs from the top left node at position $n \geq 3$ to the bottom right node:

$$f(n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \times \mathbf{T}^{n-1} \times \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

In general, when a transition matrix is diagonalizable into $\mathbf{T} = \mathbf{S}^{-1}\mathbf{J}\mathbf{S}$ for some matrix $\mathbf{S}$ and diagonal matrix $\mathbf{J}$, the value of $\mathbf{T}^n$ can be easily calculated by exponentiating the diagonal entries in $\mathbf{J}$ since $\mathbf{T}^n = \mathbf{S}^{-1}\mathbf{J}^n\mathbf{S}$. Note that when $\mathbf{T}$ is diagonalizable, the entries in $\mathbf{J}$ will be the eigenvalues of $\mathbf{T}$ and there will be coefficients c_{ij} such that:

$$f(n) = \sum_{i=1}^{k} \left(\sum_{j=1}^{q_i} c_{ij} \cdot n^{j-1} \cdot {\lambda_i}^n \right) \tag{4}$$

where $\lambda_1, \lambda_2, \ldots, \lambda_k$ are the eigenvalues of $\mathbf{T}$ and q_i is the multiplicity of λ_i.

Suppose $\lambda_m = \max(\lambda_1, \lambda_2, \ldots, \lambda_k)$, then $n^{q_m} {\lambda_m}^n$ will be the dominating term, and the value of $f(n)$ will asymptotically grow as fast as $n^{q_m} {\lambda_m}^n$.

Equivalently, $f(n) \in \Theta(n^{q_m} \cdot {\lambda_m}^n)$, where:

$$\begin{aligned} f(n) \in \Theta(g(n)) \quad &\Leftrightarrow \quad f(n) \in \Omega(g(n)) \wedge f(n) \in O(g(n)) && (5)\\ f(n) \in \Omega(g(n)) \quad &\Leftrightarrow \quad \exists k_1, n_1, \forall n > n_1, f(n) \geq k_1 \cdot g(n) && (6)\\ f(n) \in O(g(n)) \quad &\Leftrightarrow \quad \exists k_2, n_2, \forall n > n_2, f(n) \leq k_2 \cdot g(n) && (7) \end{aligned}$$

Note that if $f(n) \in \Theta(g(n))$ and $h(n) < g(n), \forall n$, then $f(n) \in \Omega(h(n))$. Similarly, if $f(n) \in \Theta(g(n))$ and $h(n) > g(n), \forall n$, then $f(n) \in O(h(n))$.

For example, the matrix $\mathbf{T}$ above has the eigenvalues: $3 + \sqrt{5}, 3 - \sqrt{5}, 2, 1$ with 2 and 1 having the multiplicity of 2 and 4, respectively. And after solving for the c_{ij} we have:

$$f(n) = \frac{(3+\sqrt{5})^{n-1}}{4\sqrt{5}+10} - \frac{(3-\sqrt{5})^{n-1}}{4\sqrt{5}-10} - 1 \tag{8}$$

From Equation 8, we see that the function $f(n)$ grows with order $(3 + \sqrt{5}) \approx 5.24$.

Now, applying the same method to our original hypergraph, we can get the order in which the number of canonical encodings grows. For the SHARED model with at most two components, we have the following transition matrix:

$$\mathbf{T}_{\text{SHARED}_2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 3 & 1 & 5 & 0 & 3 & 1 & 5 \\ 1 & 0 & 2 & 0 & 5 & 0 & 6 & 0 \\ 1 & 2 & 2 & 4 & 5 & 10 & 6 & 12 \\ 0 & 1 & 3 & 5 & 0 & 5 & 11 & 17 \\ 0 & 3 & 3 & 21 & 0 & 15 & 11 & 73 \end{bmatrix}$$

with the maximum eigenvalue $\lambda_m \approx 80.61$ with multiplicity of 1. Therefore the number of possible DAGs in the SHARED model with two components, which is the number of canonical encodings, is in the order of $\Omega(80^n)$. Analogously, we have the largest eigenvalue for the transition matrix for the SHARED model with three components to be: $\lambda_m \approx 1261.86 > 1024 = 2^{10}$ with multiplicity of 1, and so $\mathcal{M}_{\text{SH}}(n) \in \Omega(2^{10n})$.

Now, by the definition of Ω, for some n_0, we have for all $n > n_0$: $\mathcal{M}_{\text{SH}}(n) \geq C \cdot 2^{10n}$ for some constant $C > 0$ and so $\forall n > n_0, \log \sum_{i=1}^{n} \mathcal{M}_{\text{SH}}(i) > \log \mathcal{M}_{\text{SH}}(n) \geq \log C + \log 2^{10n} = \log C + 10n \log 2$.

Now we can calculate the relative ambiguity between the linear-chain model and our SHARED model as:

$$\mathcal{A}_r(\text{LI}, \text{SH}) = \lim_{n\to\infty} \frac{\log \sum_{i=1}^{n} \mathcal{M}_{\text{SH}}(i)}{\log \sum_{i=1}^{n} \mathcal{M}_{\text{LI}}(i)} \geq \lim_{n\to\infty} \frac{\log C + 10n \log 2}{3n \log 2} = \frac{10}{3} > 1 \tag{9}$$

This means the linear-chain model is more ambiguous compared to our model.

Similarly, we can compute the growth order of our SPLIT model to arrive at the conclusion that $\mathcal{M}_{\text{SP}} \in \Omega(2^{15n})$. Since $\mathcal{M}_{\text{SH}}(n) \in O(2^{11n})$, we have $\mathcal{A}_r(\text{SH}, \text{SP}) > \frac{15}{11} > 1$.

Number of canonical encodings for small n One may be concerned that since the analysis above concerns asymptotic values, it is applicable only for large n. To address that concern, we show in Table 1 the number of canonical encodings in the linear-chain model and our models, compared with the number of possible entity combinations. Note that it is hard to calculate the number of canonical encodings for the linear-chain model, since the labels interact in a complex way. For smaller n, we can exhaustively enumerate the possible encodings, but as n increases it becomes not feasible, so for $n \geq 4$ we show the upperbound instead, which is 7^n. We will show how to calculate $\mathcal{N}(n)$, the number of all possible entity combinations, later.

n	$\mathcal{M}_{\text{LI}}(n)$	$\mathcal{M}_{\text{SH}}(n)$	$\mathcal{M}_{\text{SP}}(n)$	$\mathcal{N}(n)$
1	2	2	2	$2^1 = 2$
2	8	8	8	$2^3 = 8$
3	46	80	80	$2^7 = 128$
4	< 2401	3584	6656	$2^{15} = 32768$
5	< 16807	533504	2367488	$2^{31} = 2147483648$

Table 1: The number of possible encodings for small values of n

For $n = 1, 2$, the number of possible entity combinations is small, and we see that all models can accurately represents each of them. However, note that our models have the advantage of not having the possibility to output non-canonical encodings for $n = 1, 2$, unlike the linear-chain model, which has 7 and 49 possible encodings for $n = 1, 2$, respectively. For $n = 3$, exhaustive enumeration of the encodings for the linear-chain model shows that there are 46 canonical encodings, which is less than the 80 canonical encodings in our models. For $n \geq 4$, which is typical for sentence length, we see that our models have more canonical encodings compared to the linear-chain model, which is bounded above by 7^n, although they are still smaller than the total number of possible entity combinations.

Calculating $\mathcal{N}(n)$ Now we show how to calculate the number of possible entity combinations in a sentence of length n. In the paper [Lu and Roth, 2015], the authors had established that the number of possible entity combinations when there is no discontiguous entity is $2^{t\frac{n(n+1)}{2}}$, where t is the number of possible entity types. For this case, we set $t = 1$ as there is only one entity type in this task and use the combinatorial identity $\binom{n+1}{2} = \frac{n(n+1)}{2}$ to have $2^{\binom{n+1}{2}}$ as the number of contiguous entities.

For the number of discontiguous entities with exactly k components, there is one start position and one end position for each of the k components, and all of them are distinct positions. So the number of discontiguous entities is the number of ways to choose the $2k$ positions from $n + 1$ positions, resulting in $\binom{n+1}{2k}$ possible discontiguous entities with exactly k components. The total number of discontiguous entities with at most k components is then $N = \sum_{i=1}^{k} \binom{n+1}{2i}$. Theoretically, each of these entities can exist independently of each other, resulting in possibly overlapping entities. Considering the overlapping entities, in total we have:

$$\mathcal{N} = 2^N \tag{10}$$

distinct entity combinations with each discontiguous entities having at most k components.

Note that as k goes to $\frac{n+1}{2}$ (which means when we do not restrict the number of components an entity can have), the formula above reduces to 2^{2^n-1} by the combinatorial identity $\sum_{i=0}^{\lfloor \frac{n+1}{2} \rfloor} \binom{n+1}{2i} = 2^n$, which matches the interpretation that when we do not restrict the number of components, then any non-empty subset of words can be an entity, for which there are 2^n-1 of them. Then since we count entity combinations, we have 2^{2^n-1} combinations, which is what we got when we let k goes to $\frac{n+1}{2}$.

In our case, since we are calculating the number of all possible entity combinations for $n \leq 5$, the maximum number of components an entity can have is 3, so the number of all possible entity combinations is also the number of entity combinations with at most 3 components, which is also what we calculated for the models.

2 Data Preprocessing

We performed some preprocessing on the clinical texts, which are: sentence splitting, tokenization, and POS tagging.

Splitting and tokenization We used the document preprocessor from Stanford CoreNLP package to split the documents into sentences, then further processed the output using some rules to better capture the structure of the document. We then tokenized each sentence using a regex-based tokenization, similar to NLTK `wordpunct_tokenize` function. We again further processed the output to handle special anonymization tokens (*e.g.*, "[**doctor first name 77**]") and normalized them into categories (*e.g.*, "doctor_name"). After tokenization, we ran Stanford POS tagger on the resulting text to get the POS tags for each token. Finally, we assigned each disorder mention into the corresponding sentence that contains it.

Note that in this process there might be disorder mentions which do not fit into any sentence. This happens when the sentence splitter split two words that are annotated as part of a single entity. In our case, we found that there is only one entity in the training set which is incorrectly annotated to also include the whitespace preceeding a sentence, which is not part of any sentence after sentence splitting. We fixed the annotation by excluding the preceding whitespace.

Determining section names The clinical texts in the datasets are semi-structured, in the sense that the contents are organized into sections. However, the section names seem to be quite irregular, having quite a number of variants of the section names with the same meaning. For the purpose of determining the section name during feature extraction, we used a regular expression to capture lines in the clinical texts that end with a colon or coming after a new line, with some heuristics to handle special cases found in the datasets. More details can be read in our code at `http://statnlp.org/research/ie/#discontiguous-mention`

3 Handling Ambiguity

Both the linear-chain CRF model and our models are still ambiguous to some degree, so during the decoding process we need to handle the ambiguity to produce the mentions. For all models, we define two general heuristics: ENOUGH and ALL. The ENOUGH heuristic handles ambiguity by trying to produce a minimal set of entities which encodes to the one produced by the model, while ALL heuristic handles ambiguity by producing the union of all possible entity combinations that encode to the one produced by the model.

More specifically, for the linear-chain CRF baseline, we first converted the labeled words into a sequence of component types: *contiguous*, *body*, and *head*, referring to the sequence of words encoded by {**B, I**}, {**BD, ID**}, and {**BH, IH**}, respectively. In the ALL heuristic we form all possible combinations between the components, considering the compatibility between components (*e.g.*, a *body* component cannot be paired with a *contiguous* component).

In the ENOUGH heuristic, each *head* component forms at most two entities by pairing it with *body* components starting from the closest ones from its left, unless there is only one remaining *body* component of that type after this process, in which case the *head* component will form the third entity by being paired with this last *body* component. After this, each unpaired *body* components of the same type will be paired up to form entities with two or three components. In [Tang et al., 2013] the authors mentioned that they used a few simple rules to convert labels to entities, but the exact details on the rules were not made available.

For our models, for ALL, we generate all possible sets of mentions encoded in the entity-encoded hypergraph by traversing all possible paths in the hypergraph, while in ENOUGH, we generate as many mentions as required to cover all edges present in the hypergraph.

4 Regularization Hyperparameter

For the experiments, the regularization hyperparameters for each setting are noted in Table 2. Note that the same regularization hyperparameter is used for both the ALL and ENOUGH heuristics during testing.

	SMALL		LARGE	
	Train-Disc	Train-Orig	Train-Disc	Train-Orig
LI	0.25	0.125	0.125	0.25
SH	0.5	0.125	0.125	0.125
SP	2.0	0.125	0.5	0.25

Table 2: Regularization hyperparameter λ for each setting

5 Splitting the Dataset

We note that the dataset that we have only contains one type of entity. In order to evaluate the models on multiple types, we create another dataset where we split the entities based on the entity-level semantic category.

Each entity in the dataset was annotated either with its Concept Unique Identifier (CUI), or with the string "CUI-less". The CUI is a number referencing certain entry in the Unified Medical Language System (UMLS) Metathesaurus. In UMLS, each entry has a corresponding semantic type, which is organized in a hierarchy. There are two major roots in the hierarchy, which are type `A` and type `B`. These two types, together with the CUI-less entities, make up the three entity types that we used in creating the dataset with multiple types.

6 Efficiency in Handling Multiple Entity Types

Theoretically, our models can handle multiple entity types more efficiently compared to the linear-chain CRF model. This is because the time complexity of our models are linear in terms of number of entity types, while it is quadratic for the linear-chain CRF model.

To see this empirically, we randomly split the entities in the dataset into n types, where $n = 1, 2, 4, 8, 16$, and took note of the time taken for training the models. We show the time per iteration relative to the time taken for handling only one type in Table 3. We can see that the time per training iteration in the baseline model increases faster than that of our model's, confirming the theoretical time complexity of the models.

#Types	1	2	4	8	16
Linear CRF	1.0	2.015	5.018	12.174	35.313
SHARED	1.0	2.067	4.552	9.455	19.272

Table 3: The time taken per training iteration for the baseline linear-chain CRF model (Linear CRF) and our SHARED model, relative to the time taken for handling one type

References

[Lu and Roth, 2015] Lu, W. and Roth, D. (2015). Joint Mention Extraction and Classification with Mention Hypergraphs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP'15)*, pages 857–867.

[Muis and Lu, 2016] Muis, A. O. and Lu, W. (2016). Learning to Recognize Discontiguous Entities. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP '16).*

[Tang et al., 2013] Tang, B., Wu, Y., Jiang, M., Denny, J. C., and Xu, H. (2013). Recognizing and Encoding Discorder Concepts in Clinical Text using Machine Learning and Vector Space. In *Proceedings of the ShARe/CLEF Evaluation Lab.*