Problem Statement

Let's get acquainted with the task of building an entity linking system.

We'll cover the following

- Introduction
- Applications
- Problem statement
- Interview questions

Introduction

Named entity linking (NEL) is the process of detecting and linking entity mentions in a given text to corresponding entities in a target knowledge base.

There are two parts to entity linking:

1. *Named-entity recognition*

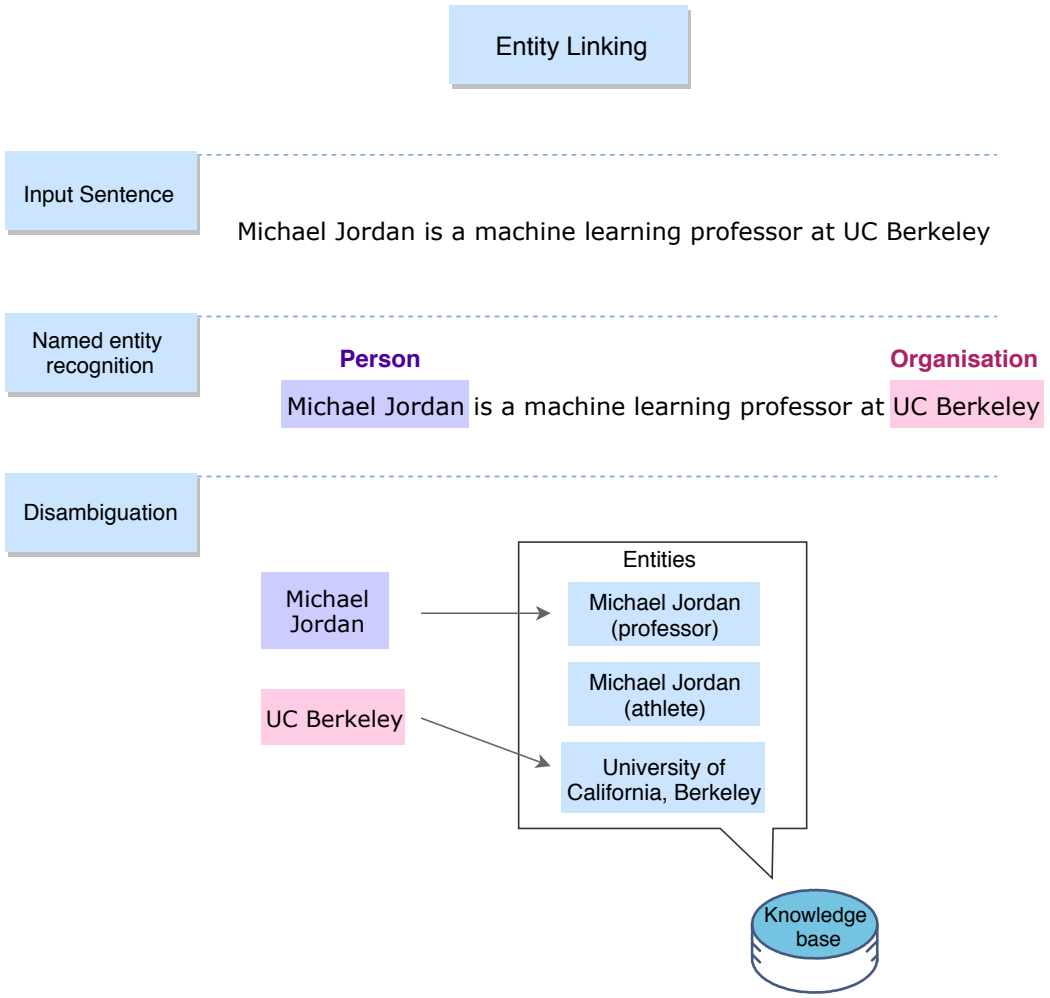
   Named-entity recognition (NER) detects and classifies potential named entities (https://en.wikipedia.org/wiki/Named_entity) in the text into predefined categories such as a person, organization, location, medical code, time expression, etc. (multi-class prediction).

2. *Disambiguation*

   Next, disambiguation disambiguates each detected entity by linking it to its corresponding entity in the knowledge base.

> 📝 The target knowledge base depends on the application, but for generic systems, a common choice is Wikidata or DBpedia.

Let’s see entity linking in action in the following example.



Entity linking

The sentence/text says, “Michael Jordan is a machine learning professor at UC Berkeley.” First NER detects and classifies the named entities Michael Jordan and UC Berkeley as person and organisation.

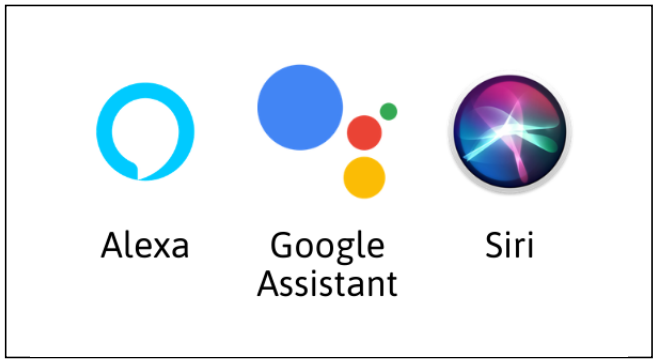
Then disambiguation takes place. Assume that there are two ‘Michael Jordan’ entities in the given knowledge base, the UC Berkeley professor and the athlete. Michael Jordan in the text is linked to the *professor at the University of California, Berkeley* entity in the knowledge base (that the text is referring to). Similarly, UC Berkeley in the text is linked to the *University of California* entity in the knowledge base.

Applications

Entity linking has applications in many natural language processing tasks. The use cases can be broadly categorized as information retrieval, information extraction and building knowledge graphs, which in turn can be used in many systems, such as:

- Semantic search
- Content analysis

- Question answering systems/chatbots/virtual assistants, like Alexa, Siri, and Google assistant



NEL applications. Image source: www.alexeko.com

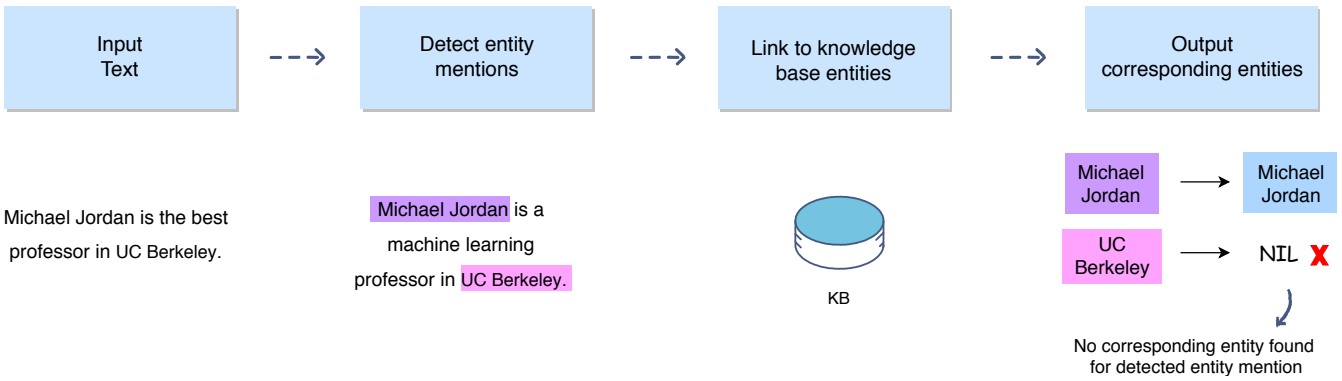
All of the above-mentioned applications require a high-level representation of the text, in which concepts relevant to the application are separated from the text and other non-meaningful data.

Now that we have introduced named entity linking and its applications, we are ready to look at the problem statement.

Problem statement

The interviewer has asked you to design an entity linking system that:

- Identifies potential named entity mentions in the text.
- Searches for possible corresponding entities in the target knowledge base for disambiguation.
- Returns either the best candidate corresponding entity or nil.



Design a named entity linking system

The problem statement translates to the following machine learning problem:

"Given a text and knowledge base, find all the entity mentions in the text(Recognize) and then link them to the corresponding correct entry in the knowledge base(Disambiguate).""

Interview questions

These are some of the questions that an interviewer can put forth.

1. How would you build an entity recognizer system?
2. How would you build a disambiguation system?
3. Given a piece of text, how would you extract all persons, countries, and businesses mentioned in it?
4. How would you measure the performance of a disambiguator/entity recognizer/entity linker?
5. Given multiple disambiguators/recognizers/linkers, how would you figure out which is the best one?

Back

Modeling

Next

Metrics

Metrics

Let's go over the metrics to evaluate the performance of the entity linking system.

We'll cover the following

- Offline metrics
  - Named entity recognition
  - Disambiguation
  - Named-entity linking component
    - Micro vs. macro metrics
- Online metrics

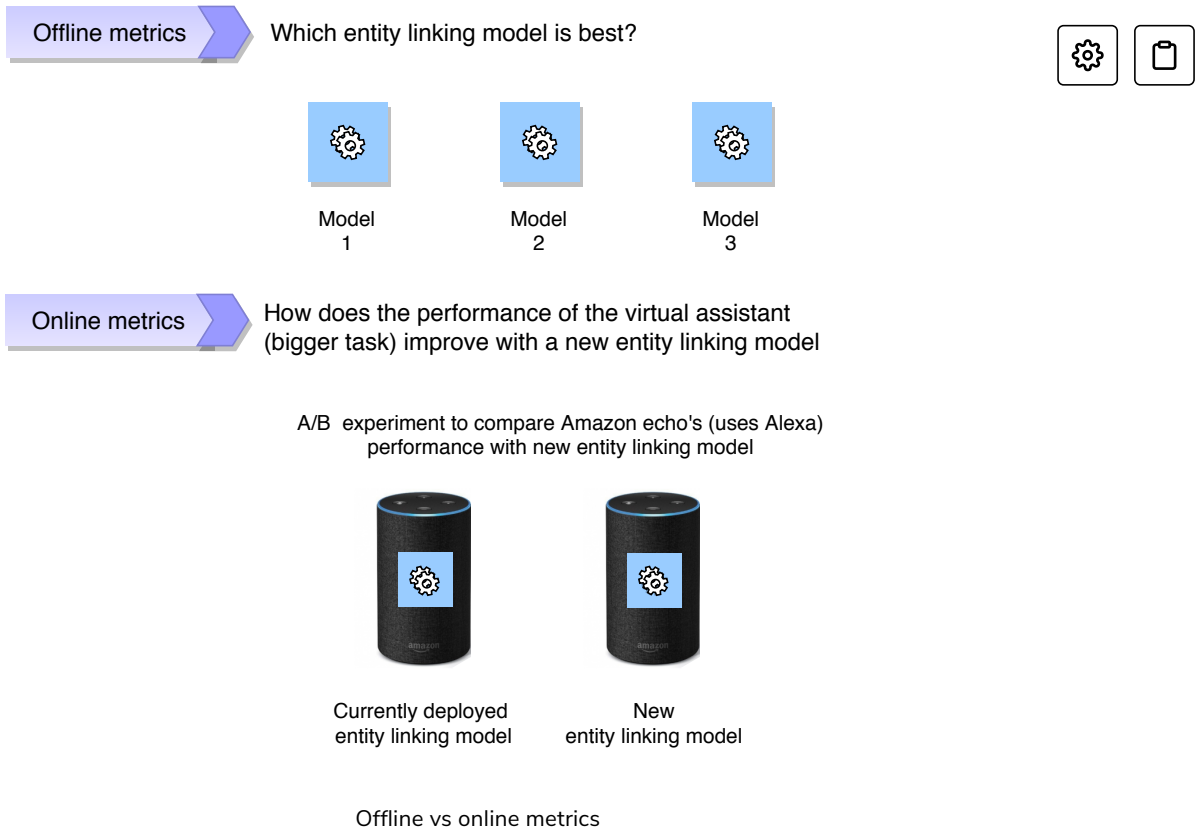
In the previous lesson, we talked about various applications of *named entity linking* system and how it can be used as a component in bigger tasks/systems such as a virtual assistant system. Therefore, we require metrics that:

1. Compare different entity linking models based on their performance.

   This will be catered to by offline metrics.

2. Measure the performance of the bigger task when a particular model for entity linking is used.

   This will be catered to by online metrics.



Offline vs online metrics

> 📝 Offline metrics will be aimed at improving/measuring the performance of the entity linking component. Online metrics will be aimed at improving/measuring the performance of the larger system by using a certain entity linking model as its component.

Offline metrics

The named-entity linking component is made of two layers, as discussed previously:

1. Named entity recognition
2. Disambiguation

We will first look at offline metrics for each of the layers individually and will then discuss a good offline metric to measure the overall entity linking system.

Named entity recognition

For the first layer/component, i.e., the recognition layer, you want to extract all the entity mentions from a given sentence. We will continue with the previous sentence example, i.e., “Michael Jordan is the best professor at UC Berkeley”.

It has two entity mentions:

1. Michael Jordan

2. UC Berkeley

NER should be able to detect both entities correctly. However, it may detect:

- Both correctly
- One correctly
- None correctly (wrongly detect non-entity as an entity)
- Correct entity but with the wrong type
- No entity, i.e., altogether miss the entities in the sentence

Possible NER predictions

	Sentence	Michael	Jordan	is	the	best	professor	at	UC Berkeley
	Label	B-Person	I-Person	O	O	O	O	O	B-Organisation
	Possible predictions by NER								
Both entities correctly detected	1.	B-Person	I-Person	O	O	O	O	O	B-Organisation
One entity correctly detected	2.	O	O	O	O	O	O	O	B-Organisation
	3.	B-Person	I-Person	O	O	O	O	O	O
No entity detected	4.	O	O	O	O	O	O	O	O
Wrong detection	5.	O	B-Organ-isation	O	O	O	B-Person	O	O

Note: this diagram is made using **IOB2** (aka BIO) tagging scheme for NER

Key

B: Beginning of entity
I: inner token of a multi-token entity
O: non-entity

Possible predictions by NER for given sentence

> 📝 You will call a recognition/detection of a named entity correct, only if it is an exact match of the entity in the labeled data. If NER only recognizes “Michael” as an entity and misses the “Jordan” part, it would be considered wrong. Moreover, if NER recognizes “Michael Jordan” as an entity but with the wrong type (say Organization), again, it would be considered wrong.

Given the above context on the correctness of the system, both precision and recall are important for measuring the performance of NER. They will be defined as:

Precision $= \frac{no.\ of\ correctly\ recognized\ named\ entities}{no\ of\ total\ recognized\ named\ entities}$

Recall $= \frac{no.\ of\ correctly\ recognized\ named\ entities}{no\ of\ named\ entities\ in\ corpus}$

You may have models that have excellent precision but poor recall. For instance, when such a model recognizes entities, they will be mostly correct. However, it may not recognize all the entities present in the sentence. Similarly, you could have models that have excellent recall but poor precision.

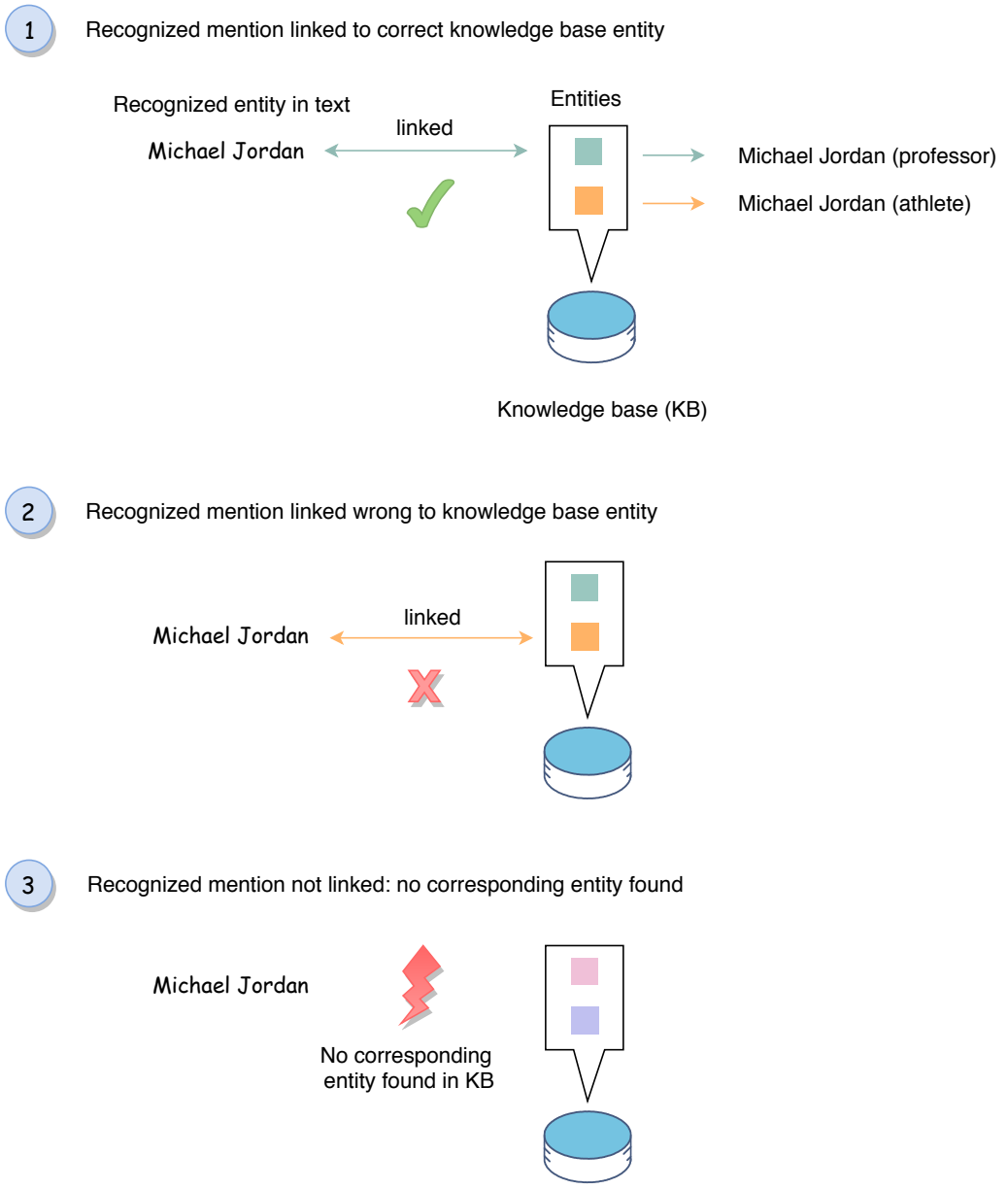
You want your model to have both high precision and high recall. To look at both, collectively, you will use the F1-score as the metric.

F1-score = $2 * \frac{precision * recall}{precision + recall}$

Disambiguation

The disambiguation layer/component receives the recognized entity mentions in the text and links them to entities in the knowledge base. It might:

- Link the mention to the correct entity
- Link the mention to the wrong entity
- Not link the mention to any entity (in case it does not find any corresponding entity in the knowledge base)



Possible disambiguation outputs for recognized entity mentions in text

This component will perform linking for all entities recognized, and they will either be linked to an object in the knowledge base or not linked at all (i.e., the model predicts that it doesn't have a corresponding object in the knowledge base). So, the concept of recall doesn't really apply here as each entity is going to be linked(to either an object or Null). Therefore, it makes sense to only use precision as the metric for disambiguation.

$$\text{Precision} = \frac{no.\ of\ mentions\ correctly\ linked}{no.\ of\ total\ mentions}$$

Named-entity linking component

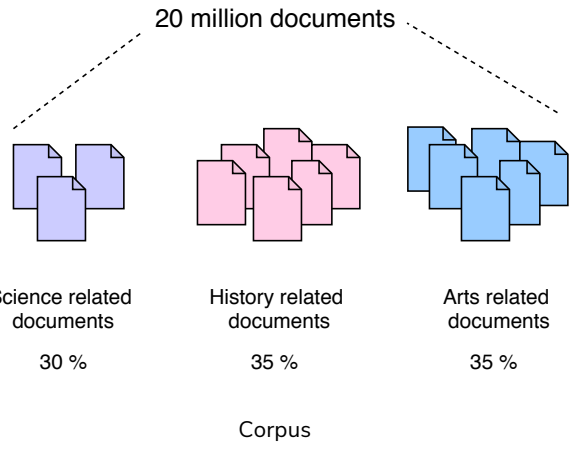
You have seen the metrics for two main components of the entity linking system. Now, let's devise a metric to measure the goodness provided by the entity linking component as a whole. You will use F1-score as the end-to-end metric. The definition of a true positive, true negative, false positive, and false negative, for the calculation of end-to-end F1-score (and precision and recall by extension), is as follows:

- *True positive:* an entity has been correctly recognized and linked.
- *True negative:* a non-entity has been correctly recognized as a non-entity or an entity that has no corresponding entity in the knowledge base is not linked.
- *False positive:* a non-entity has been wrongly recognized as an entity or an entity has been wrongly linked.
- *False negative:* an entity is wrongly recognized as a non-entity, or an entity that has a corresponding entity in the knowledge base is not linked.

Micro vs. macro metrics

We can compute *micro-averaged* or *macro-averaged* versions of metrics, based on what is important for us in a particular situation.

Let's take an example of a corpus of documents to see scenarios where micro and macro metrics will have significantly different results. Let's assume that the corpus has one million documents that contain twenty million entities collectively. The documents can be categorized as science (30% of the corpus), arts (35% of the corpus), and history (35% of the corpus) related.

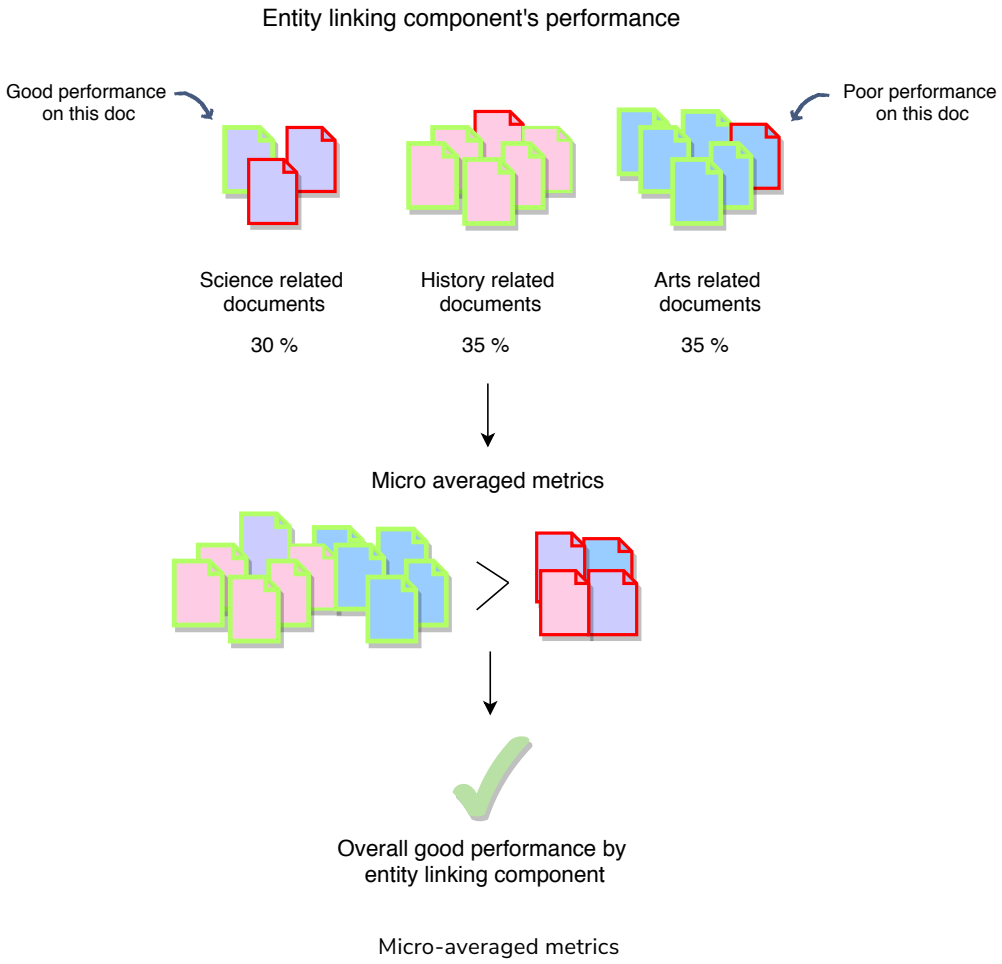


> A macro-average computes the metric independently for each document and takes the average (giving equal weightage to all documents). In contrast, a micro-average aggregates the contributions of all documents to compute the average metric.

- Micro-averaged metrics

Assume that you are only interested in how well you recognize these twenty million entities, without paying any attention to the performance across individual documents. Here, you will opt for micro-averaged F1-score and micro-averaged precision and recall, by extension.

> 📝 We are focused on the overall performance of the entity linking component. We don't care if the performance is better for a certain set of documents and not good for another set, so we opt for micro-averaged metrics.



Micro-averaged metrics

Micro-averaged metrics are computed, as shown below.

Micro-averaged precision = $\frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + \sum_{i=1}^{n} FP_i}$

where n*=no. of documents,* TP_i*=true positives in document* $'i'$ *and* FP_i*=false positive in document* $'i'$

Micro-averaged recall = $\frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + \sum_{i=1}^{n} FN_i}$ *where* FN_i*=false positive in document* $'i'$

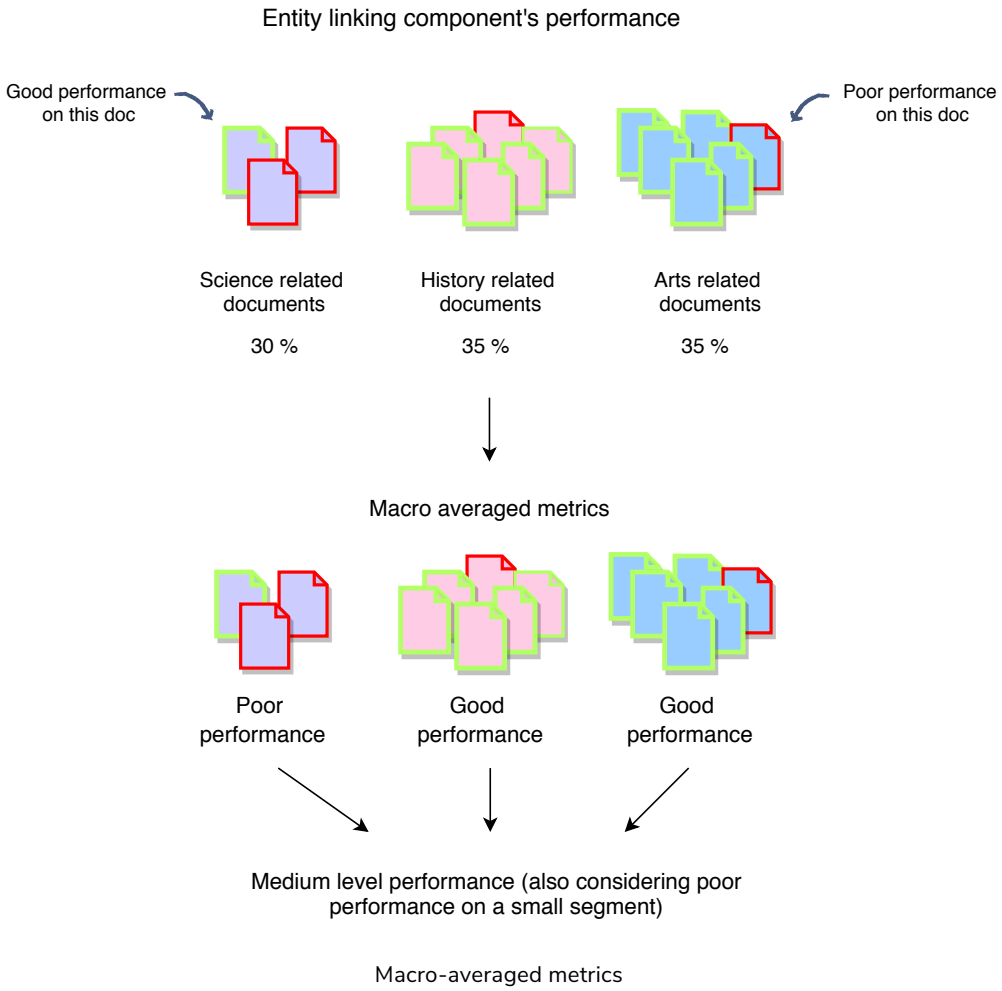
Micro-averaged F1-score = $2 * \frac{Micro-averaged\ precision * Micro-averaged\ recall}{Micro-averaged\ precision + Micro-averaged\ recall}$

- Macro-averaged metrics

When you are interested in the individual performance of entity linking across the different types of documents (e.g., science, history, and arts), you will shift to macro-averaged f1 score and macro averaged precision and recall, by extension.

Let's look at a scenario where you would need to look at entity linking's performance across different types of documents. Assume that 30% of science-related documents contain only 10% of the entities. Now entity linking performs poorly for this small segment, but it performs well for the rest of the 70% documents. Macro averaged metrics allow you to calculate the metric for each set of documents and then take their average, thereby giving an equal chance of representation to the small segment of science-related documents.

On the contrary, micro-averaged metrics would not pay attention to these small number of documents where performance is low and give a biased result.



Macro-averaged metrics are computed, as shown below.

Macro-averaged precision = $\frac{\sum_{i=1}^{n} P_{di}}{n}$ $where\ n{=}no.\ of\ documents,\ P_{di}{=}precision\ over\ document\ 'i'$

Macro-averaged recall = $\frac{\sum_{i=1}^{n} R_{di}}{n}$ $where\ n{=}no.\ of\ documents,\ R_{di}{=}recall\ over\ document\ 'i'$

Macro-averaged F1-score = $2 * \frac{Macro-averaged\ precision * Macro-averaged\ recall}{Macro-averaged\ precision + Macro-averaged\ recall}$

Online metrics

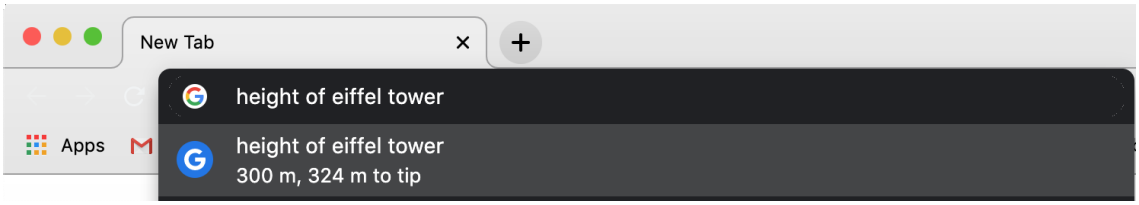
Once you have a model with good offline scores, you still need to see if the bigger task/system's performance will improve if you plug in your new model. So, you would do A/B experiments for these bigger systems to measure their performance with your new entity linking component. The metrics in these A/B tests would be devised based on the overall system, which, in turn, would indicate how well your new model for entity linking is performing as it gets integrated.

To get a better understanding, let's think about two such larger tasks/systems:

1. Search engines
2. Virtual assistants

Search engines

Semantic search (https://en.wikipedia.org/wiki/Semantic_search) allows us to directly answer the user's query by returning the entity or its properties that the user wants to know. The user no longer needs to open up search results and look for the information that is required. As shown in the example below, the user typed their query in the search bar, where entity linking recognized and linked the entity mention *"eiffel tower"*. This enables the system to fetch the entity's property "height" from the knowledge base and directly answer the user's query.



Entity linking for search query allows us to provide direct answers

You have seen how a search engine may use your entity linking component. Now, let's come up with the evaluation metric for the search engine.

For search engines, user satisfaction lies in the query being properly answered, which can be measured by *session success rate*, i.e., % of sessions with user intent satisfied. So, your online A/B experiment will measure the effect of your new entity linking systems on the session success rate.

Virtual assistants

Virtual assistants (https://en.wikipedia.org/wiki/Virtual_assistant) (VAs) helps perform tasks for a person based on commands or questions. For instance, a user asks Alexa, "What is the height of the Eiffel tower?". In order to answer this question, the VA needs to detect and link the entities in

the user's question (the entity linking component is required here). In this case, the entity would be "Eiffel tower". Once this has been done, the VA can fetch the entity's property "height" and answer the user's question.

The evaluation metric for the VA would be user satisfaction (percentage of questions successfully answered). It could be measured in various ways by using implicit and explicit feedback. However, the key aspect here is that user satisfaction should improve by plugging in a better model for the entity linking component in the system.

Back

Problem Statement

Next

Architectural Components

Mark as Completed

69% completed, meet the criteria and claim your course certificate!

Report an Issue

Ask a Question
(https://discuss.educative.io/tag/metrics__entity-linking-system__grokking-the-machine-learning-interview)

Architectural Components

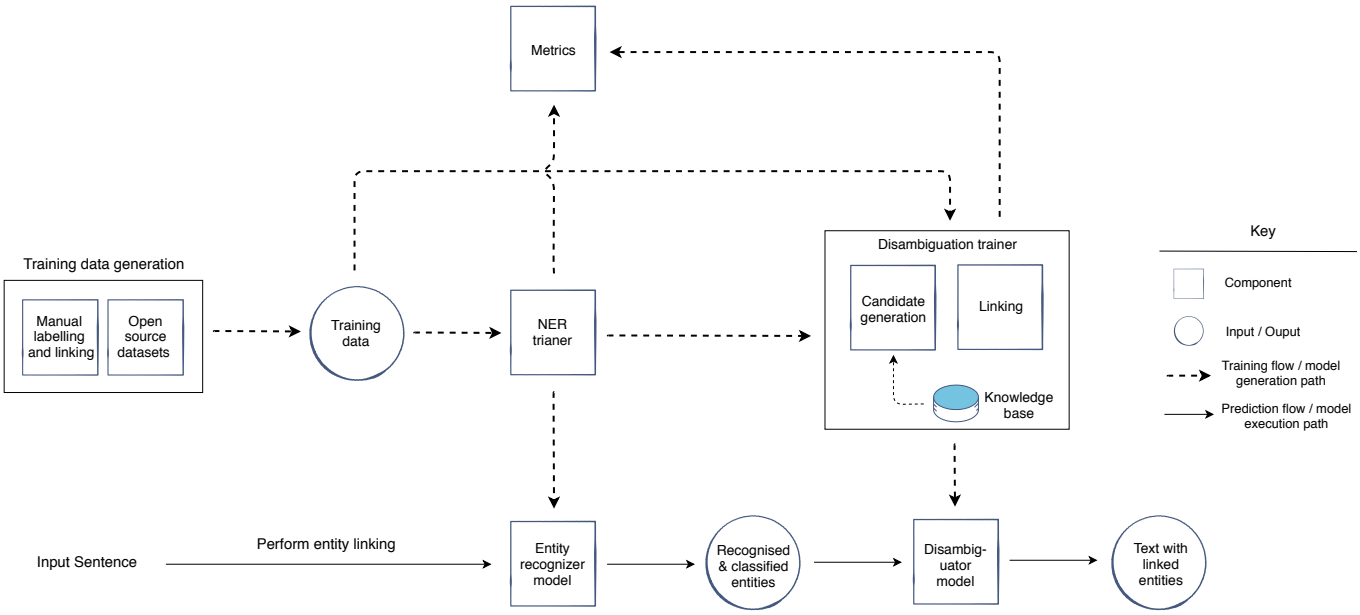
Let's go over the architectural components of the entity linking unit.

We'll cover the following

- Model generation path
  - Training data generation
  - NER
  - NED
  - Metrics
- Model execution path

The architectural components diagram for entity linking is shown below. It consists of two paths:

1. Model generation path (training flow)
2. Model execution path (prediction flow)



Architectural diagram for entity linking

📝 The terms *entity mention* and *recognized entity* are used interchangeably.

Model generation path

Model generation is responsible for training models for entity linking task. Let's look at the components of this path.

Training data generation

You will begin by gathering training data for entity linking through open-source datasets and manual labelling/linking of text. These methods will be detailed in the next lesson.

You will pass the training data to the named entity recognition (NER) model trainer and the named entity disambiguation (NED) model trainer.

NER

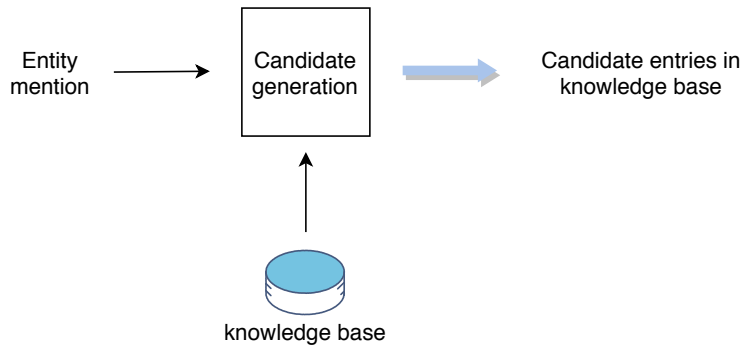
NER is responsible for building a machine learning model to recognize entities, such as a person, organization, etc., for a given input text.

NED

The disambiguation component will receive the output of the NER for linking. The disambiguation process has two phases:

1. Candidate generation

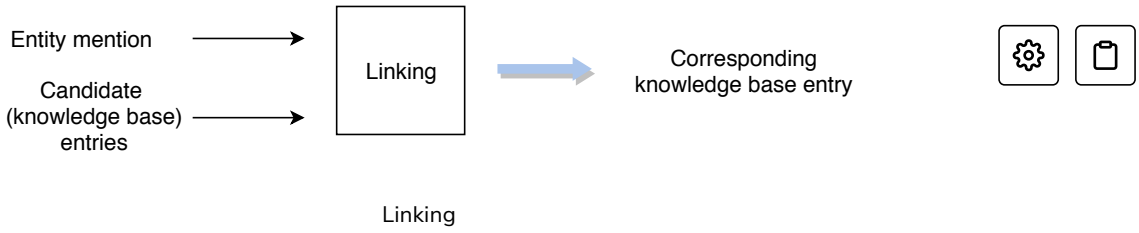
   The first phase of disambiguation is candidate generation. It finds potential matches for the entity mentions, by reducing the size of the knowledge base to a smaller subset of *candidate documents/entities*. This saves us from running the linking model on the entire knowledge base for each entity mention.



Candidate generation

2. Linking

   The second phase of disambiguation is linking. Here, you will select the exact corresponding entry in the knowledge base for each recognized entity. The linking model runs on only the candidate entries for each mention.



Linking

Metrics

As explained in the previous lesson, the metrics component will measure the performance of:

1. NER component separately
2. NED component separately
3. Entity linking as a whole

> 📝 The arrow going from disambiguation to the metrics component shows that the metrics component will not only measure the quality of the disambiguation but the quality of the whole entity linking system.

Model execution path

The model execution path is very straight forward. It begins with an input sentence that is fed to the NER component. NER identifies the entity mentions in the sentence, along with their types, and sends this information to the NED component. This component then links each entity mention to its corresponding entity in the knowledge base (if it exists). Like this, entity linking will be performed for a given piece of text.

Back
Metrics

Next
Training Data Generation

Training Data Generation

Let's generate training data for the entity linking problem.

We'll cover the following

- Open-source datasets
- Human-labeled data

There are two approaches you can adopt to gather training data for the entity linking problem.

1. Open-source datasets
2. Manual labeling

You can use one or both depending on the particular task for which we have to perform entity linking.

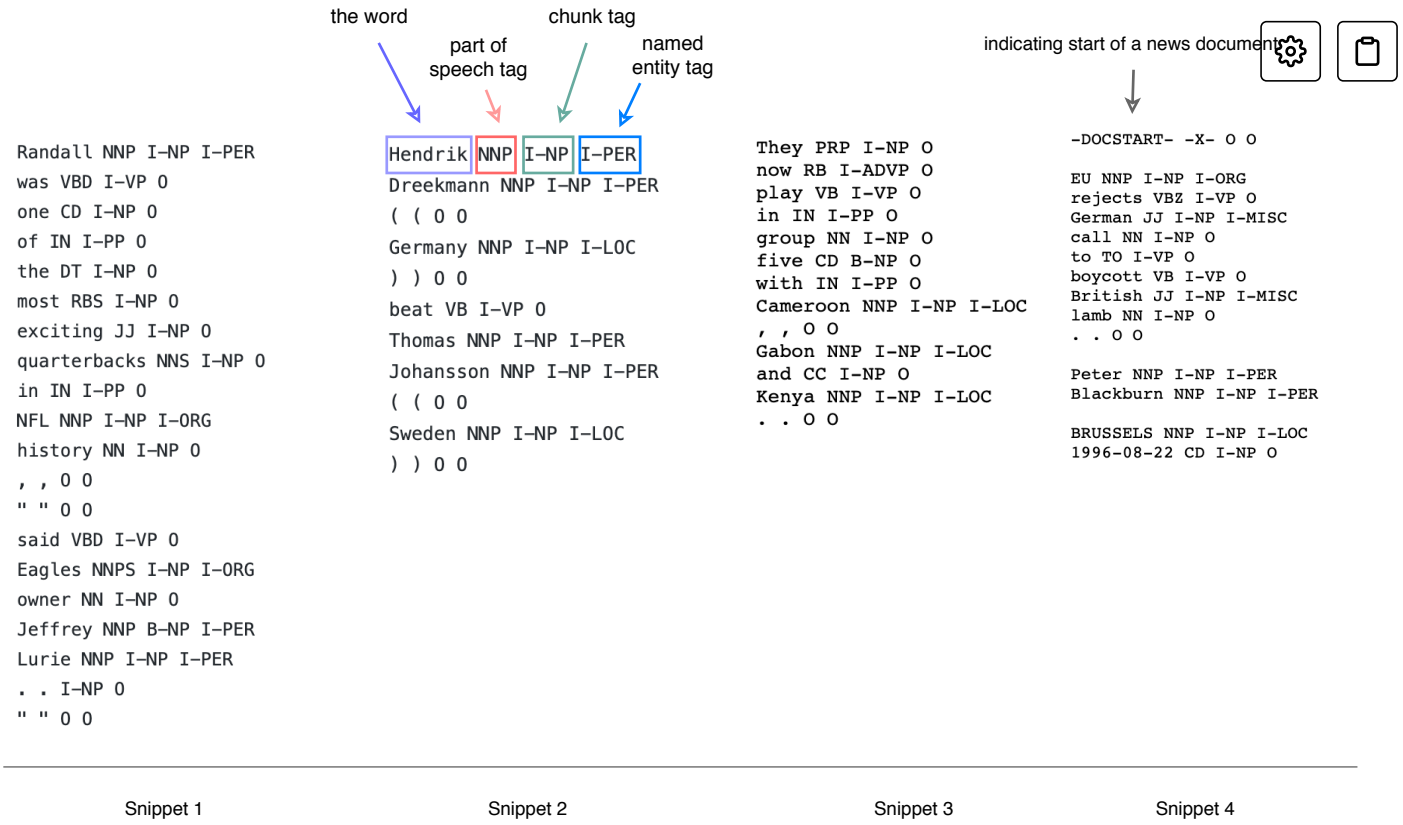
Open-source datasets

If the task is not extremely domain-specific and does not require very specific tags, you can avail open-source datasets as training data. For example, if you were asked to perform entity linking for a simple chatbot, you could utilize the general-purpose, open-source dataset CoNLL-2003 (https://www.clips.uantwerpen.be/conll2003/ner/) for *named-entity recognition*.

CoNLL-2003 is built on the Reuters Corpus which contains 10,788 news documents totalling 1.3 million words. It contains train and test files for English and German languages and follows the IOB tagging scheme.

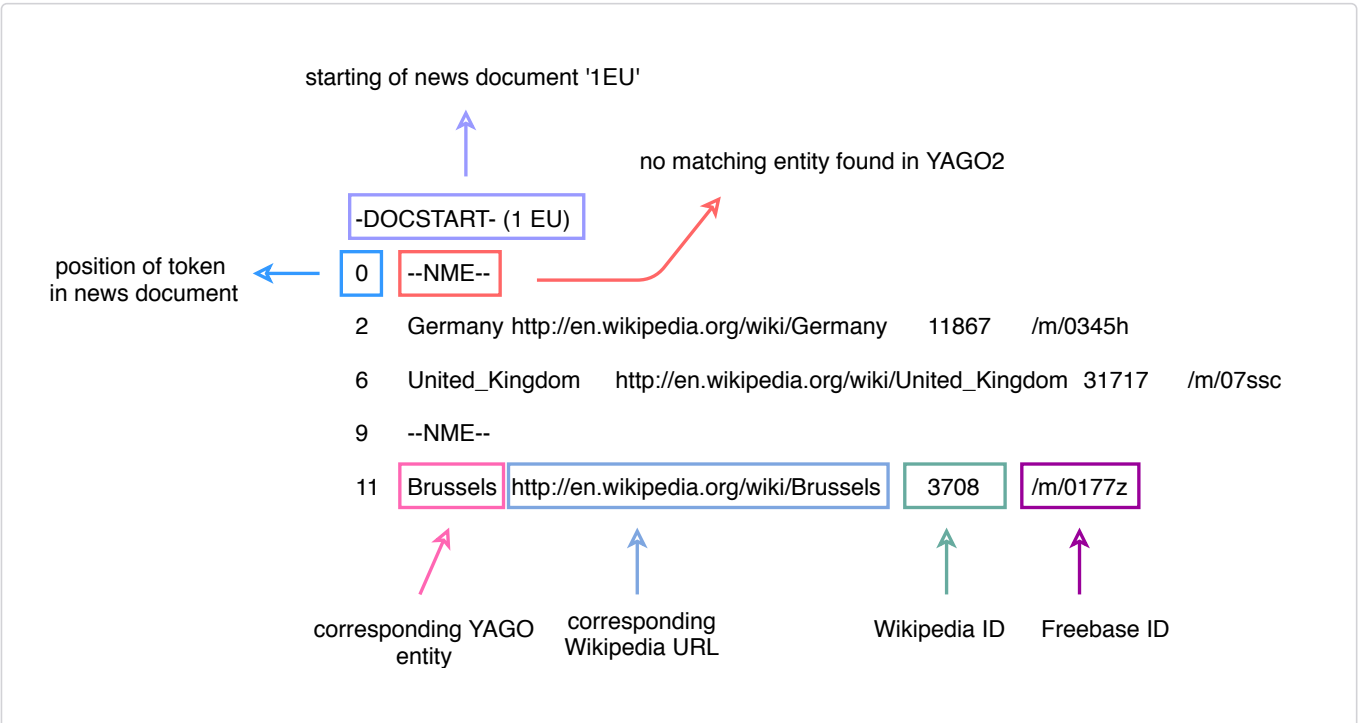
> 📝 IOB tagging scheme
> `I` - An inner token of a multi-token entity
> `O` - A non-entity token
> `B` - The first token of a multi-token entity; The B-tag is used only when a tag is followed by a tag of the same type without “O” tokens between them. For example, if for some reason the text has two consecutive locations (type LOC) that are not separated by a non-entity

The following are some snippets from the train and test files of CoNLL dataset.



CoNLL-2003 for named entity recognition

For *named-entity disambiguation*, you can utilize the AIDA CoNLL-YAGO Dataset (https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/aida/downloads/), which contains assignments of entities to the mentions of named entities annotated for the CoNLL-2003 dataset. The entity mentions are assigned to YAGO (database), Wikipedia and Freebase (database) entities as shown in the slide below.



Human-labeled data

Once you have utilized the open-source datasets, we may want to enhance the data and increase its size through manual labelers. The manual labelers will generate training data similar to the open-source datasets, by annotating named entities in text and linking them to corresponding entities in the knowledge base.

Another case where you would generate data through manual labelers is when you require a highly specialized dataset for a specific problem. For example, assume that the problem is related to the medical field; this requires identifying certain domain-specific entities. In such situations, you need to understand the domain in which you want to perform entity linking. What are the kind of entities you want to recognize and link? When the manual labelers are given hospital data, they will mark doctor names, symptoms, diseases, patient names, types of surgeries, and so on. Hence, you would have tags that are related to the domain of the task.

After labeling the entities the labelers will also link them to the entities in the knowledge base (database) that is being used.

Back
Architectural Components

Next
Modeling

Mark as Completed

Modeling

Let's look at the modeling options for entity linking.

We'll cover the following

- Contextualized text representation
  - ELMo
  - BERT
- NER modelling
  - Contextual embedding as features
  - Fine-tuning embeddings
- Disambiguation modeling
  - Candidate generation
  - Linking

The first step of entity linking is to build a representation of the terms that you can use in the ML models. It’s also critical to use contextual information (i.e., other terms in the sentence) as you embed the terms. Let’s see why such representation is necessary.

Contextualized text representation

It is often observed that the same words may refer to a different entity. The context (i.e., other terms in the sentence) in which the words occur help us figure out which entity is being referred to. Similarly, the NER and NED models require context to correctly recognize entity type and disambiguate, respectively. Therefore, the representation of terms must take contextual information into account.

One way to represent text is in the form of embeddings. For instance, let’s say you have the following sentences:

1 Michael Jordan is the best professor at UC Berkeley.

2 Regarded by most as the NBA's greatest all-time player, Michael Jordan resides in Florida.

The words "Michael Jordan" refer to different people in the two sentences

When you generate the embedding for the words “Michael Jordan”, through traditional methods such as Word2vec, the embedding would be the same in both sentences. However, you can see that, in the first sentence, “Michael Jordan” is referring to the UC Berkeley professor. Whereas, in the second sentence, it is referring to the basketball player. So, the embedding model needs to consider the whole sentence/context while generating an embedding for a word to ensure that its true meaning is captured.

Notice that, in the first sentence, the context that helps to identify the person comes after the mention. Whereas, in the second sentence, the helpful context comes before the mention. Therefore, the embedding model needs to be bi-directional, i.e., it should look at the context in both the backward direction and the forward direction.

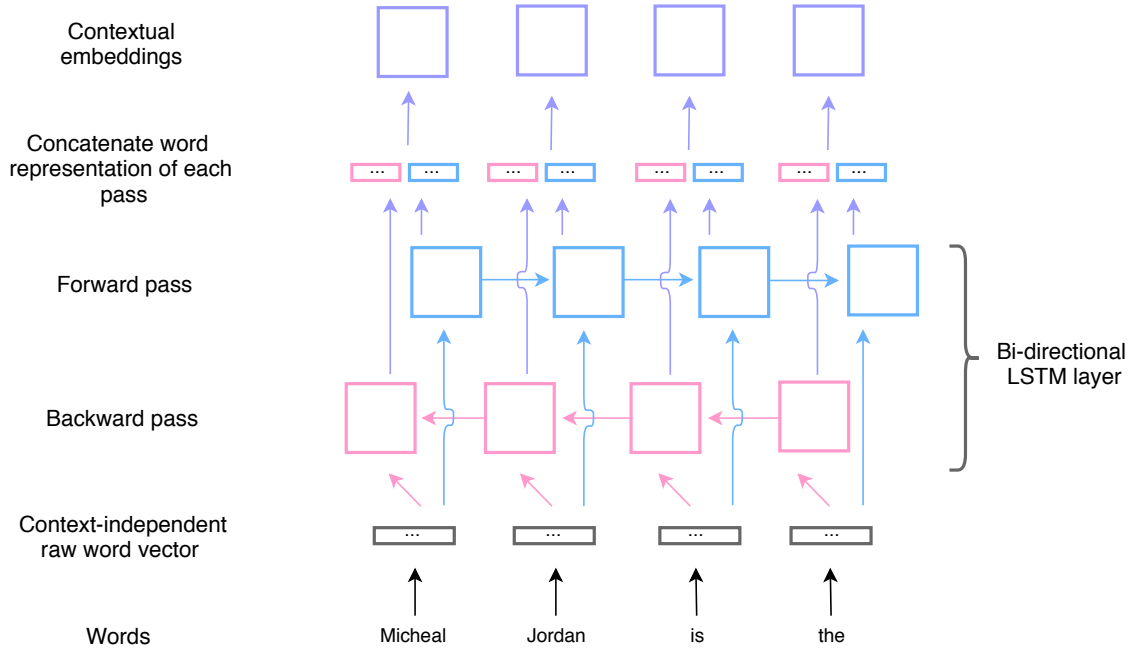
Two popular model architectures that generate term contextual embeddings are:

1. ELMo
2. BERT

ELMo

Let’s see how ELMo (Embeddings from Language Models) generates contextual embeddings. It starts with a character-level convolutional neural network (CNN) or context-independent word embedding model (e.g., Word2vec) to represent words of a text string as raw word vectors. The raw vectors are fed to a bi-directional LSTM layer trained on a language modeling (*generating the next word in a sentence conditioned on previous words*) objective. This layer has a forward pass and a backward pass.

The forward pass sequentially goes over the input sentence from left to right. It looks at the context (words) before the active word. Whereas, the backward pass sequentially goes over the input sentence from right to left. It looks at the context (words) after the active word to predict the current word. Contextual information from both these passes is concatenated and then combined in another layer to obtain contextual embeddings of the text.

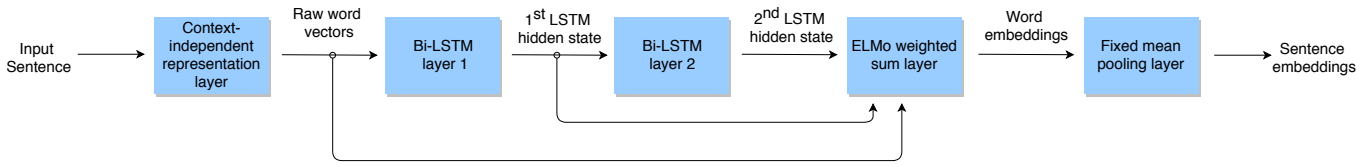


ELMo (unravelled)

📝 If the raw word vector is made using a character level CNN, the inner structure of the word is captured. For instance, the similarity in the structure of the words “learn” and “learning” will be captured, which will be helpful information for the bi-directional LSTM layer.

The character level CNN will also make good raw vectors for out-of-vocabulary words by looking at their similarity with the vocabulary observed during the training phase.

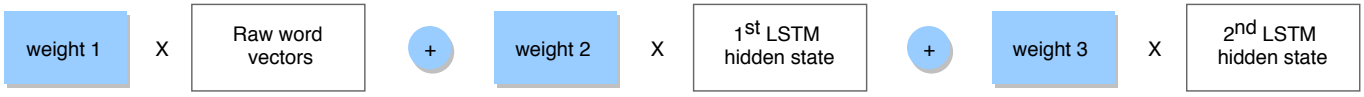
ELMo can be used to obtain *both word embeddings and sentence embeddings*. Below is a common implementation of ELMo, based on this paper (https://arxiv.org/pdf/1802.05365.pdf).



A common implementation of ELMo

📝 Adding more layers allows the model to learn even more context from the input. The initial layers help identify grammar and syntactic rules while the deeper layers help extract higher contextual semantics.

The input sentence is fed to the context-independent representation layer. It outputs raw word vectors which are passed on to the first Bi-LSTM layer. The hidden state/ intermediate word vectors from this layer are fed to the second Bi-LSTM layer. It also outputs a hidden state/intermediate word vectors. The ELMo weighted sum layer performs a weighted sum of the outputs of the three previous layers to arrive at the word embeddings.



ELMo weighted sum layer

> 📝 The weights shown in the above diagram are trainable.

The fixed mean pooling layer takes the word embeddings and converts them into a sentence embedding.

> 📝 The previous diagram, labeled “ELMo (unravelled)”, zooms in on the context-independent and first Bi-LSTM layer. The intermediate word vectors made by this Bi-LSTM layer can also be taken as wording embeddings.

Although ELMo uses bi-directional LSTM, it is a *“shallow” bi-directional model*. It is deemed “shallow” because of how it achieves bi-directionality. There is a sequential pass on the input from:

- Left to right to train the forward LSTM
- Right to left to train the backward LSTM

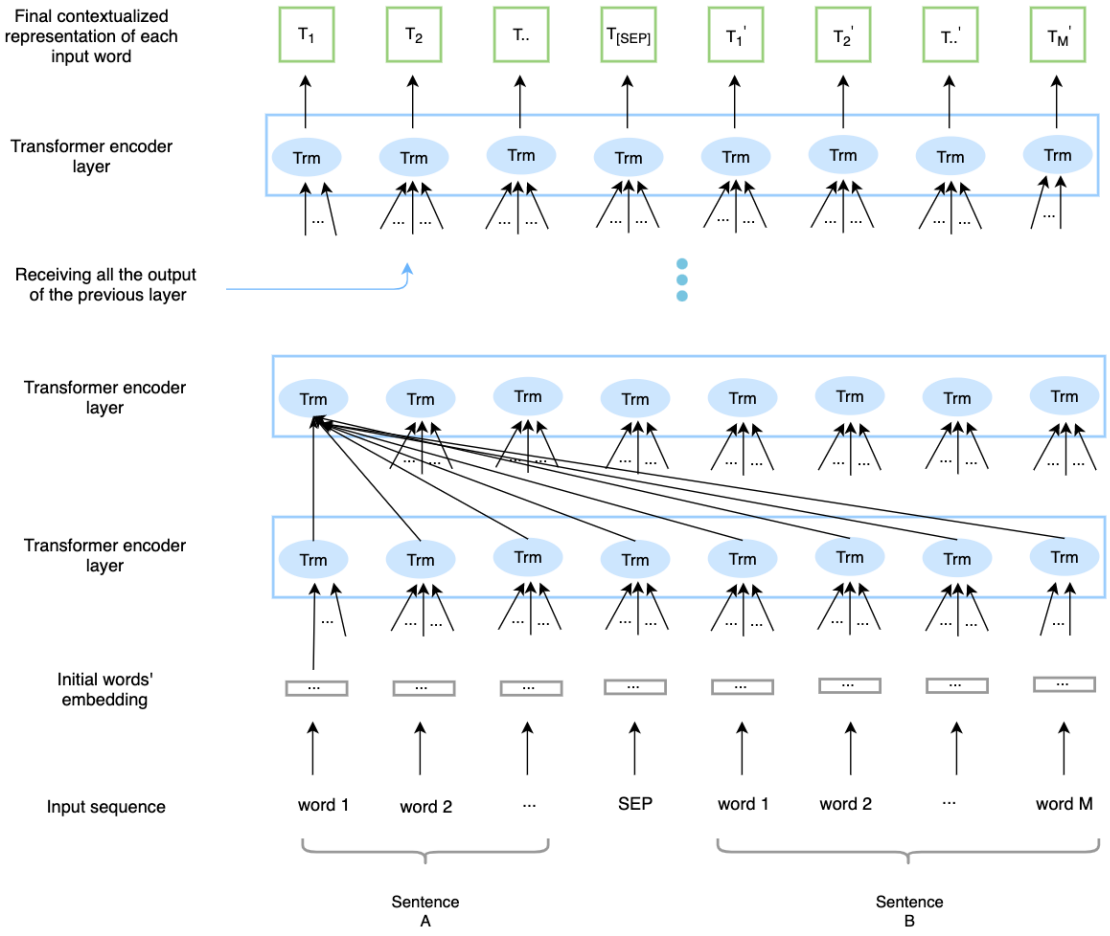
The forward and backward LSTMs are trained *“independently”*, and only their word representations are concatenated. Therefore, the word representations can’t take advantage of both the left and right contexts *“simultaneously”*.

BERT

Let’s see how BERT (bidirectional encoder representations from transformers) gives us contextual word embeddings.

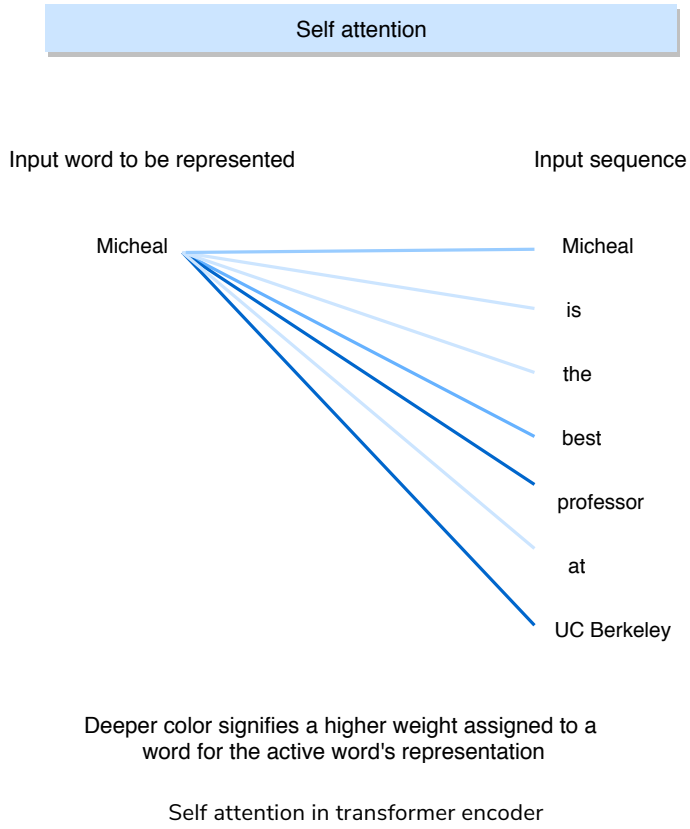
BERT starts by taking in the input sequence which can be made up of multiple sentences separated by the *SEP* tag. Each word is converted into embeddings and fed to the first transformer encoder layer. This layer has the capability to process all the words of the input sequence simultaneously. Therefore, the word representations produced are based on context from both directions, jointly.

The output of this transformer layer is passed onto the next layer, and this happens for all the transformer layers. The final transformer layer outputs the contextualized representation of each word.



BERT

The transformer encoder block uses the concept of *self-attention* while computing the representation for the words in an input sequence. Simply put, self-attention is the process of assigning weights to the words in the input sequence according to their relevance/contribution to the understanding of the word whose representation is being made.



You see that the words: “professor” and “UC Berkeley”, followed by “best”, are given more weights in the representation of the word “Michael”. These words help the system to understand the active word better.

The following two prediction objectives are used during the training of the model for learning contextualized term representation:

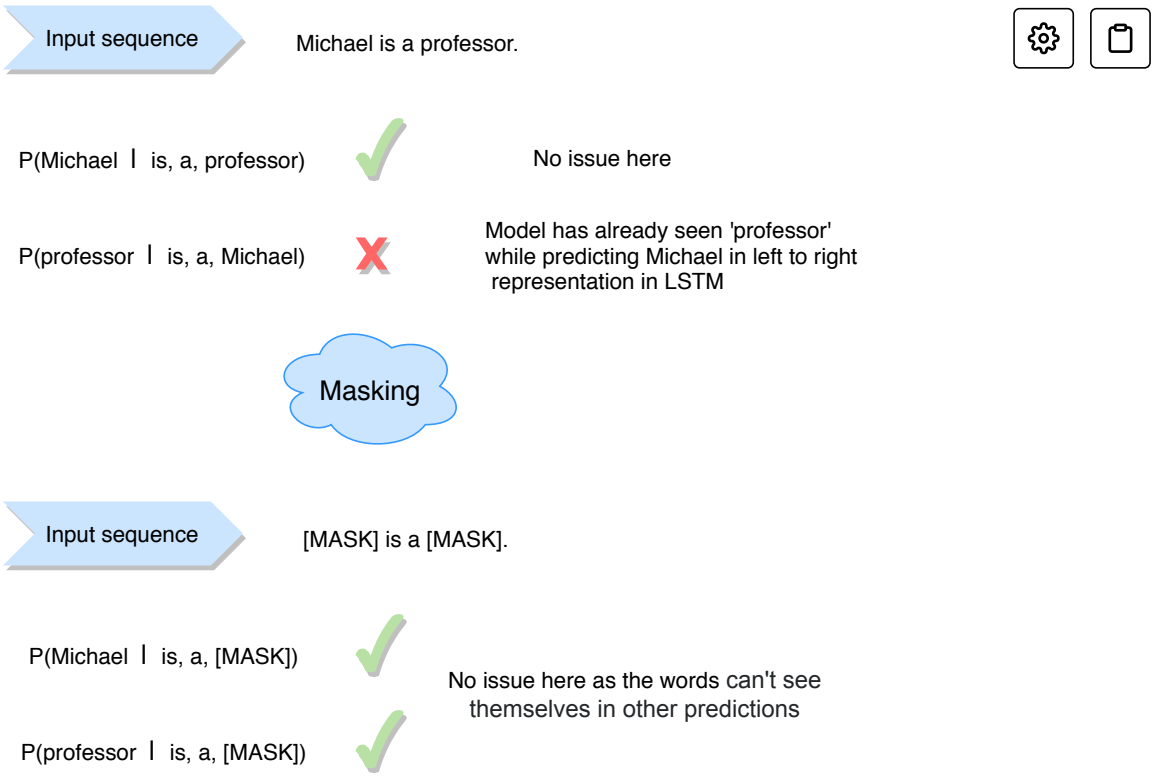
1. Masked language modeling (MLM)

   A few words are masked in the input, and the model predicts them based on the bi-directional context.

2. Next sentence prediction (NSP)

   Two sentences A and B are given. The model has to predict if B comes after A in the corpus or is it just a random sentence.

Let’s quickly see how these objectives make BERT a *deeply bi-directional model,* in contrast to ELMo. You will use the MLM objective as an example.



How masking alleviates the problem with bi-directional context in BERT, which you couldn't do earlier with Bi-LSTM

BERT takes the input sequence in one go as compared to sequential input. The problem with looking at the bidirectional context is that the word that's being predicted indirectly "sees itself". It appears in the bi-directional context for the prediction of another word in the input sequence. Masking helps hide some words so that this problem is alleviated.

> 📝 ELMo has solves this problem through sequential prediction from left to right and right to left, independently. This is suboptimal because it results in shallow bi-directionality.

Variations

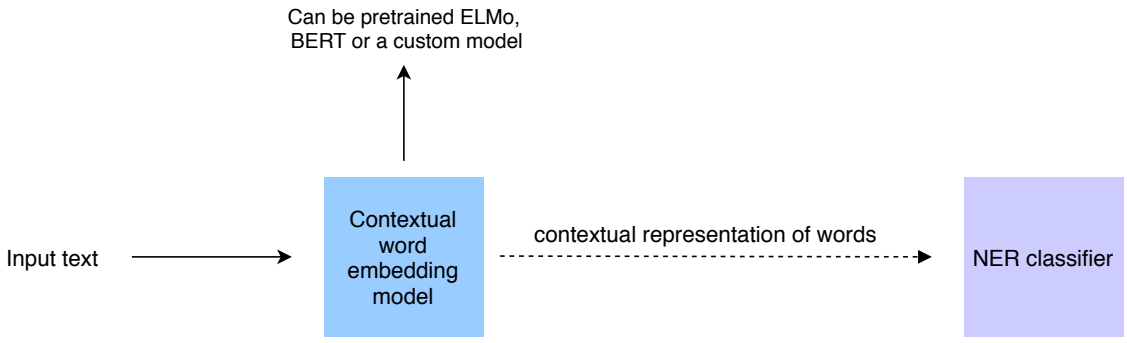
Let's look at some variations of BERT.

BERT Variation	Description
BERT base	This is a smaller and faster version. It has 12 layers/transformer blocks and 110 million parameters.

BERT Variation	Description
BERT large	This is the larger version. It has 24 layers and 340 million parameters.
DistilBERT	If execution time is a concern, we have versions of BERT that are faster than BERT base. They squeeze the network by smaller embeddings, fewer layers, or some other methods. One such version is called *DistilBERT*. With a slight compromise on the quality, it retains 97% of its language understanding capabilities and is 60% faster. **It makes sense to use DistilBERT to embed tokens when evaluation time performance is critical.**
Cased vs uncased BERT	BERT has a cased and uncased variation. **It makes sense to use the cased version knowing that case is important in NER** (generally the upper case is used when writing names).

NER modelling

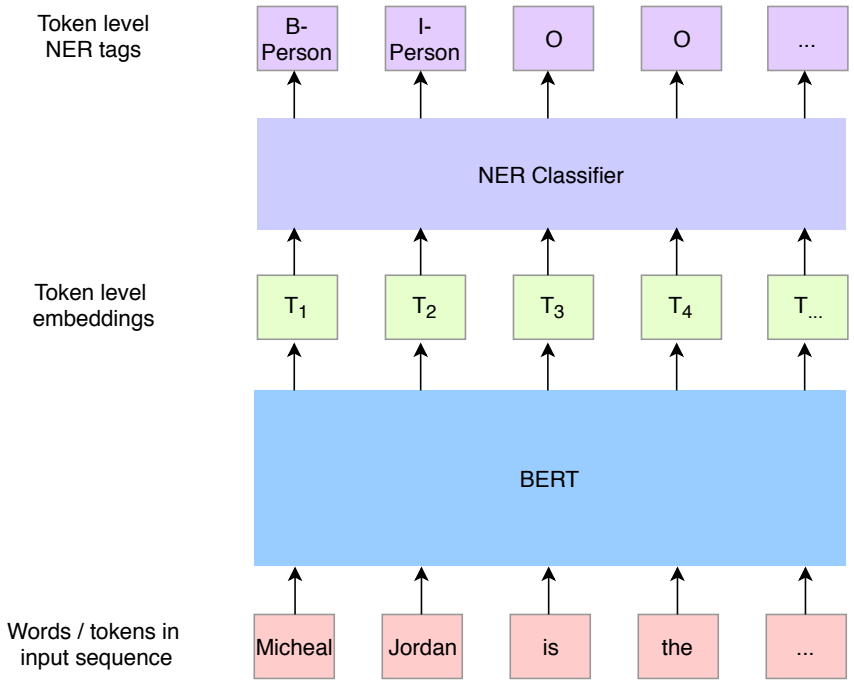
Both ELMo and BERT are trained on massive datasets and have the ability to understand language. With these two models generating contextual word embeddings, half of the work is done. Through transfer learning approach, you can now utilize these embeddings in a NER classifier.



There are two methods to utilize pre-trained models for the task:

Contextual embedding as features

One quick way to utilize these contextual embeddings generated by BERT is to use them as features in your NER modelling.



Once BERT gives us token level embeddings, you can train a classifier on top of token embeddings to predict NER classes.

Fine-tuning embeddings

Another option is to take the pre-trained models generated on a large corpus (e.g., BERT base, BERT large, and DistilBERT) and fine-tune them based on your NER dataset to improve the classifier quality. This makes more sense, especially when we have large NER labelled data set. For BERT fine-tuning, we can either fine-tune the whole model or only top k layers, depending on how much training data you have and also on training time (performance).

> 📝 If the interviewer asks you to not use these large pre-trained models due to time or resource constraints, you can build your own customized model based on similar concepts.

Disambiguation modeling

Once you have identified the entity mentions in the text through NER, it's time to link them to corresponding entries in the knowledge base. As mentioned in the architectural components lesson, the disambiguation process consists of two phases:

1. Candidate generation

In this phase, for each recognized entity, you will select a subset of the knowledge-base entities as candidates that might correspond to it.

2. Linking

   In this phase, for each recognized entity, you will select a corresponding entity from among the candidate entities. Thanks to candidate generation, you will only have to choose from a smaller subset instead of the whole knowledge base.

Candidate generation

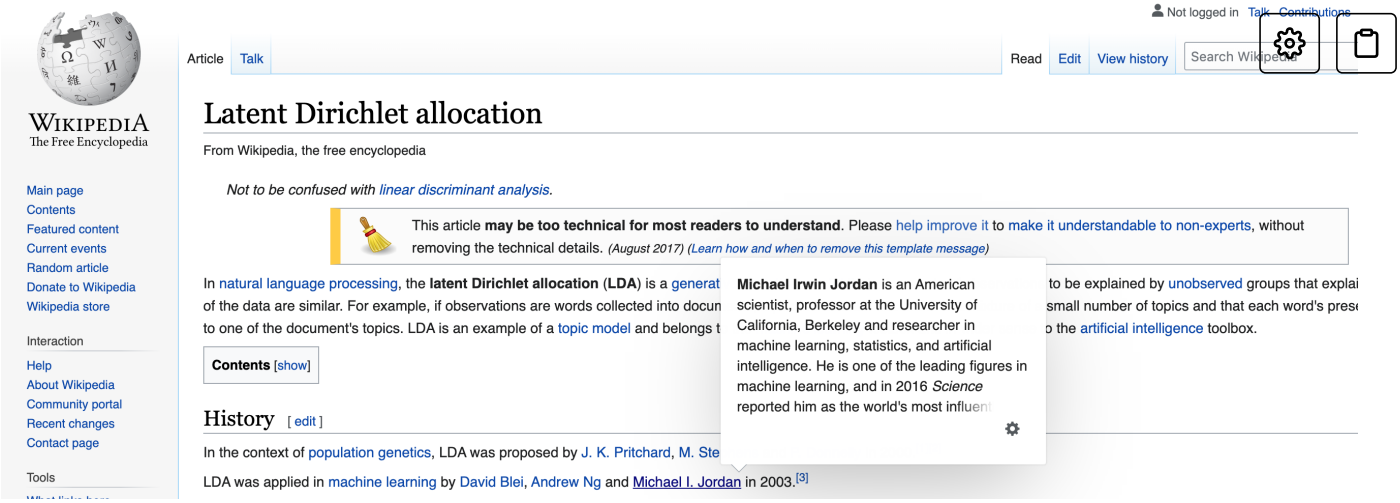
In candidate generation, you will look up the terms in the recognized entities in an index to retrieve candidate entities. So, the process of candidate generation requires building an index where terms are mapped to knowledge base entities. This index will help us in the quick retrieval of candidates for recognized entities. To build this index, you need *ways* to figure out what terms should be used to index each entity in the knowledge base.

Before you start looking at such *ways*, it is important to point out that you need to build the index such that *candidate generation focuses on higher recall*. The index should include all terms that could possibly refer to an entity, even if it is something as trivial as a nickname or a less frequently used term for an entity. The reason behind it is that you do not want to shorten the list of candidates for the linking stage at the cost of missing out on potential matches (one of which could have been the true match).

Now let's look at some of the ways to look for terms to index each entity on.

1. You will index a knowledge base entity on all the terms in its name and their concatenations. For instance, for the entity "Michael Irwin Jordan", the index terms can include "Michael Irwin Jordan", "Michael Jordan", "Michael", "Irwin", "Jordan", "Jordan Michael", and so on. Now, if you encounter any of these terms in the text, you can say that they might be referring to the entity "Michael Irwin Jordan" in the knowledge base.

2. You can also make use of referrals and anchor text (https://en.wikipedia.org/wiki/Anchor_text) in the knowledge-base data for this purpose. For instance, assume that an anchor text reads "Michael I. Jordan" and refers to the "Michael Irwin Jordan" entity in the knowledge-base. Here, you will index the entity on the terms in the anchor text as well. This way, you will get a flavour of a lot of different ways in which a knowledge base entity may be referred to in text such as nicknames and abbreviations.



The document on LDA has anchor text "Michael I. Jordan" that links to the "Michael Irwin Jordan" entity

3. If you are provided with a source that can give us commonly used spellings, aliases, and synonyms for an entity's name, then you can also use these terms for indexing. However, if that is not the case, then you can use the embedding method.

   Embedding method

   You know about some terms that link to a particular entity by methods such as the two described above. In order to discover more terms to index a particular entity on, you can look for words that are similar to the ones an entity is already indexed on.

   The first step to finding similar words is representing all the words in the knowledge base with the help of embeddings. You can use a pre-built embedding model or build one ourselves. The model will try to bring the abbreviation/ aliases/ synonyms, that refer to the same entity closer in the embedding space.

   Once you have the embedding of all the words, you can find k nearest neighboring terms for a particular term that is already linked to an entity. These k nearest neighboring terms can also be used to index the entity.

Linking

Once you are done generating candidates from the knowledge base entities for the recognized entity mentions in the given input sentence, you need to perform linking.

In the linking stage, you will build a model that will give us the probability of a candidate being the true match for a recognized entity. You will select the candidate with the highest probability and link it to the recognized entity mention.

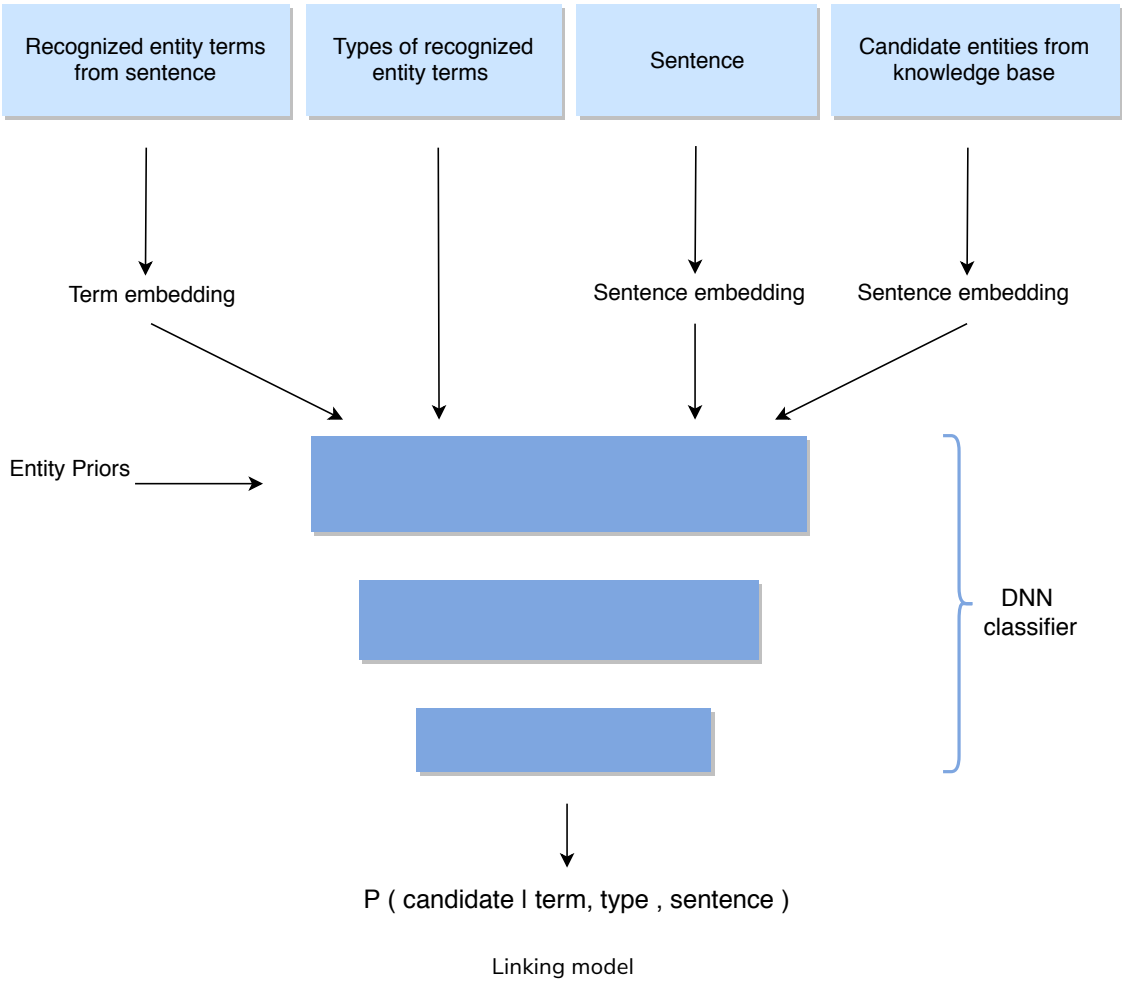
Let's look at the linking model's inputs. It will receive:

- the recognized entity mention that you want to link

- the type of the entity mention, e.g., location, person etc.
- the whole input sentence in which the entity mention is recognised
- the candidate entity from the knowledge base
- the prior: for a certain entity mention, how many times the candidate entity under consideration is actually being referred to. For example, the anchor text “Harrison Ford”, referred to the entity American actor (https://en.wikipedia.org/wiki/Harrison_Ford) 98% of the time instead of the silent film actor “Harrison Edward Ford”. It’s important to tell the model that priors favor a certain entity more.

> 📝 Utilization of the candidate entity in the anchor text is the “prior” in the above example.

All of these inputs will be fed to a deep neural network (DNN) classifier, which will give us the required probability.



Linking model

> 📝 This stage focuses on precision, i.e., you want to identify the correct corresponding entity for a mention.

It is important to consider how the inputs (entity mention, sentence and candidate entity) will be represented. The best representations are the ones given by contextualized embeddings, which you are generating through models such as BERT and ELMO for NER.

For example, assume that you have used BERT for NER, so you have the contextual embedding for the recognized entity. BERT can also provide sentence embeddings (along with term embeddings), so you can have the embedding for the entire input sentence. You can also generate the embeddings of the candidate entities based on their representation in the knowledge base. For instance, Wikidata has a small description of each entity in the knowledge base, which can be used to generate an embedding for the candidate entity.

These embeddings, along with the prior and entity type, will provide the model with all the information it needs to produce a good output.

Back

Training Data Generation

Next

Problem Statement

Mark as Completed

69% completed, meet the criteria and claim your course certificate!

Report an Issue

Ask a Question
(https://discuss.educative.io/tag/modeling__entity-linking-system__grokking-the-machine-learning-interview)