Document transformers

INFO

Head to Integrations for documentation on built-in document transformer integrations with 3rd-party tools.

Once you've loaded documents, you'll often want to transform them to better suit your application. The simplest example is you may want to split a long document into smaller chunks that can fit into your model's context window. LangChain has a number of built-in document transformers that make it easy to split, combine, filter, and otherwise manipulate documents.

Text splitters

When you want to deal with long pieces of text, it is necessary to split up that text into chunks. As simple as this sounds, there is a lot of potential complexity here. Ideally, you want to keep the semantically related pieces of text together. What "semantically related" means could depend on the type of text. This notebook showcases several ways to do that.

At a high level, text splitters work as following:

1. Split the text up into small, semantically meaningful chunks (often sentences).
2. Start combining these small chunks into a larger chunk until you reach a certain size (as measured by some function).
3. Once you reach that size, make that chunk its own piece of text and then start creating a new chunk of text with some overlap (to keep context between chunks).

That means there are two different axes along which you can customize your text splitter:

1. How the text is split
2. How the chunk size is measured

Get started with text splitters

The default recommended text splitter is the RecursiveCharacterTextSplitter. This text splitter takes a list of characters. It tries to create chunks based on splitting on the first character, but if any chunks are too large it then moves onto the next character, and so forth. By default the characters it tries to split on are `["\n\n", "\n", " ", ""]`

In addition to controlling which characters you can split on, you can also control a few other things:

- `length_function`: how the length of chunks is calculated. Defaults to just counting number of characters, but it's pretty common to pass a token counter here.
- `chunk_size`: the maximum size of your chunks (as measured by the length function).
- `chunk_overlap`: the maximum overlap between chunks. It can be nice to have some overlap to maintain some continuity between chunks (eg do a sliding window).
- `add_start_index`: whether to include the starting position of each chunk within the original document in the metadata.

```
# This is a long document we can split up.
with open('../../state_of_the_union.txt') as f:
    state_of_the_union = f.read()
```

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

```
text_splitter = RecursiveCharacterTextSplitter(
    # Set a really small chunk size, just to show.
    chunk_size = 100,
    chunk_overlap  = 20,
    length_function = len,
    add_start_index = True,
)
```

```
texts = text_splitter.create_documents([state_of_the_union])
print(texts[0])
print(texts[1])
```

```
    page_content='Madam Speaker, Madam Vice President, our First Lady
and Second Gentleman. Members of Congress and' metadata={'start_index':
0}
    page_content='of Congress and the Cabinet. Justices of the Supreme
Court. My fellow Americans.' metadata={'start_index': 82}
```

Other transformations:

Filter redundant docs, translate docs, extract metadata, and more

We can do perform a number of transformations on docs which are not simply splitting the text. With the `EmbeddingsRedundantFilter` we can identify similar documents and filter out redundancies. With integrations like doctran we can do things like translate documents from one language to another, extract desired properties and add them to metadata, and convert conversational dialogue into a Q/A format set of documents.