Implementation of a real-time semantic retrieval system.

David Lorenzo Alfaro

Jun 17, 2021

CONTENTS:

The increasingly overwhelming amount of available natural language motivates the pressing need to find efficient and reliable computational techniques capable of processing and analysing this type of data for the purpose of achieving human-like natural language understanding for a wide range of downstream tasks.

Over the last decade, Natural Language Processing (NLP) has seen impressively fast growth, primarily favoured by the increase in computational power and the progress on unsupervised learning in linguistics. Moreover, historically successful statistical language modeling techniques have been largely replaced by novel neural language modeling based on Deep Learning models, exhibiting an unprecedent level of natural language understanding, contributing to reduce the gap between human communication and computer understanding.

NLP is the key to solve many technological challenges. Among the large number of applications this field has, and since this dissertation has been entirely focused on the study of different strategies to encode salient information about natural language, the experimental part of this work is primarily devoted to the implementation of a semantic information retrieval system, delivering search latencies suitable for real-time similarity matching.

Of course, the examined unsupervised pretrained representations have been trained on humongous data sets, devoting to that end massive amounts of computational resources, something we do not have the access to. It is, however, not only a matter of computational power. Gathering, preparing and processing data for a model to be fine-tuned is no easy task and requires knowledge, to a great extent, of both the underlying theoretical motivations and the specific implementation. Consequently, our work is pretty much aligned with the mentality in which these strategies sit on: to directly use the pretrained models for downstream tasks.

This document contains the documentation for all experiments conducted throughout the dissertation, including a thoroughly description of the implementation of the information retrieval system and the notebooks devoted to the dataset analysis and preprocessing and the analysis and visualization of BERT contextual embeddings.

CHAPTER

ONE

SEMANTIC SEARCH (SEMANTIC_SEARCH.PY)

This class implements a semantic textual information retrieval system.

Author: David Lorenzo Alfaro.

class semantic_search.**SemanticSearch**(*corpus: pandas.core.frame.DataFrame*, *embeddings_cache_path: str = None*, *encoding_strategy='title_overview'*, *pretrained_model='paraphrase-distilroberta-base-v2'*, *model_max_seq_length=512*, *pretrained_crossencoder: str = None*, *annoy_index_cache_path: str = None*, *annoy_n_trees=576*, *_annoy_embedding_size=768*)

This class implements a semantic textual information retrieval system, which allows for advanced features like retrieve and re-rank and Approximate Nearest Neighbours search.

__init__(*corpus: pandas.core.frame.DataFrame*, *embeddings_cache_path: str = None*, *encoding_strategy='title_overview'*, *pretrained_model='paraphrase-distilroberta-base-v2'*, *model_max_seq_length=512*, *pretrained_crossencoder: str = None*, *annoy_index_cache_path: str = None*, *annoy_n_trees=576*, *_annoy_embedding_size=768*)

Constructor for a *SemanticSearch* instance.

Important: if you are willing to load the embeddings or the ANNOY index from disk you do not need to tune their respective specific parameters (they will be overlooked).

Parameters

- **corpus** (*DataFrame*) – Book corpus. It should, at least, have the following columns (with the very same names):
  - *gr_book_id*: book unique identifier.
  - *title*: book titles.
  - *authors*: book authors.
  - *overview*: book overview.
- **embeddings_cache_path** (*str, optional*) – Filepath to store the computed embeddings or load the embeddings from. Defaults to None.
- **encoding_strategy** (*str, optional*) – Encoding strategy to use. The encoding strategy must be a string containing the names of the features to include into the input of the encoder, each of them separated by an underscore ('_'). For example, if you were to use the title and the overview as the encoding strategy, *encoding_strategy* must be either *title_overview* or *overview_title*. Defaults to 'title_overview'.

- **pretrained_model** (*str, optional*) – The model *id* of a predefined *Sentence-Transformer* hosted inside a model repo on sbert.net. Defaults to 'paraphrase-distilroberta-base-v2'. Defaults to 'paraphrase-distilroberta-base-v2'.
- **model_max_seq_length** (*int, optional*) – Property to get the maximal input sequence length for the model. Longer inputs will be truncated. Defaults to 512.
- **pretrained_crossencoder** (*str, optional*) – Any model name from Huggingface Models Repository that can be loaded with AutoModel. Defaults to None.
- **annoy_index_cache_path** (*str, optional*) – Filepath to store an ANNOY index or load it from disk. Defaults to None.
- **annoy_n_trees** (*int, optional*) – Number of trees to use in the forest for ANNOY. Defaults to 576
- **_annoy_embedding_size** (*int, optional*) – Size of the embeddings, required to compute the index. Defaults to 768

property annoy_index

Getter method for *annoy_index*

Returns Current ANNOY index. If none, it attempts to create a new one with the optimal configuration (according to the experiments detailed in the dissertation document).

Return type AnnoyIndex

property crossencoder

Getter method for *crossencoder*.

Returns Current pretrained Cross-Encoder. If none, it attempts to obtain the default one.

Return type CrossEncoder

search(*query: str*, *k=5*, *k_biencoder=20*, *use_annoy=False*, *reranking=False*)

Perform semantic search.

Parameters

- **query** (*str*) – Textual query.
- **k** (*int, optional*) – Number of most relevant documents to retrieve. When using exhaustive search, the value of *k* does not affect perfomance. Complexity using ANNOY and *k* ~ corpus length will be close to O(n). Defaults to 5
- **k_biencoder** (*int, optional*) – If using retrieve and re-rank, number of documents to retrieve by the Bi-encoder and fed into the Cross-Encoder. The Cross-Encoder will return the *k* most relevant entries. *k_biencoder* must be greater or equal to *k*. Defaults to 20.
- **use_annoy** (*bool, optional*) – Use approximate search to reduce search time to approx O(log(n)). Defaults to False
- **reranking** (*bool, optional*) – Use retrieve and Re-Rank Pipeline, defaults to False

search_multiple(*queries: list*, *write_to: str = None*, ***search_options*)

Perform semantic search for several queries. Refer to the documentation of *search* method for further information.

Parameters

- **queries** (*list*) – Collection of textual queries.

- **`write_to`** (*`str, optional`*) – Path of the file in which the results of the query will be written. If the file already exists, existing data will be overwritten. Defaults to None.

`setup_annoy`(*index_cache_path: str = None*, *n_trees=576*, *embedding_size=768*)
Setup for ANNOY index. Use this method to:

- Use a precomputed ANNOY index located in *index_cache_path*.
- **Create a new ANNOY index and store it in *index_cache_path*.** if *index_cache_path* is *None*, the index will not be stored in disk.
- **Either way, the obtained ANNOY index will be used in future calls** to *search* and *search_multiple* if approximate search is chosen.
- Previous ANNOY setup is replaced upon invoking this method.

IMPORTANT: if you are attempting to load an ANNOY index from disk, there is no need to tune the remaining parameters (i.e., *n_trees* and *embedding_size*)

Parameters

- **`index_cache_path`** (*`str, optional`*) – Filepath to store the obtained ANNOY index or filepath of a precomputed ANNOY index. By default is *None*: a new ANNOY index will be created with the indicated parameters.
- **`n_trees`** (*`int, optional`*) – Number of trees to use in the forest for ANNOY, defaults to 576.
- **`embedding_size`** (*`int, optional`*) – Size of the embeddings, required to compute the index. Defaults to 768

`setup_crossencoder`(*pretrained_crossencoder='cross-encoder/stsb-distilroberta-base'*)
Set or update the Cross-Encoder to be used to re-rank the results retrieved by Bi-Encoder. We recomend using either 'cross-encoder/stsb-distilroberta-base' or any pretrained Cross-Encoder trained on MS MARCO dataset.

Parameters **`pretrained_crossencoder`** (*`str, optional`*) – Any model name from Huggingface Models Repository that can be loaded with AutoModel. Defaults to 'cross-encoder/stsb-distilroberta-base'

`test_annoy_performance`(*queries: list*, *k=5*, *verbose=True*)
Utility to test the performance of ANNOY, considering the speedup with respect to exhaustive search and the recall.

Parameters

- **`queries`** (*`list or array-like`*) – Collection of textual queries.
- **`k`** (*`int, optional`*) – Top k elements to consider in the comparison.

CHAPTER

TWO

LEXICAL SEARCH (LEXICAL_SEARCH.PY)

This class implements a textual literal information retrieval system.

Author: David Lorenzo Alfaro.

class lexical_search.**TfIdfSearch**(*corpus: pandas.core.frame.DataFrame*, *vectors_cache_path: str = None*, *encoding_strategy='title_overview'*, *pretrained_biencoder: str = None*, *embeddings_cache_path: str = None*, *biencoder_max_seq_length=512*, *pretrained_crossencoder: str = None*)

This class implements a textual literal information retrieval system, based on TF-IDF which allows for advanced features like hybrid search, combining literal and dense search (retrieve and re-rank search pipeline).

__init__(*corpus: pandas.core.frame.DataFrame*, *vectors_cache_path: str = None*, *encoding_strategy='title_overview'*, *pretrained_biencoder: str = None*, *embeddings_cache_path: str = None*, *biencoder_max_seq_length=512*, *pretrained_crossencoder: str = None*)

Constructor for a *TfIdfSearch* instance.

Important: if you are willing to load the embeddings or the vectors from disk you do not need to tune their respective specific parameters (they will be overlooked).

Parameters

- **corpus** (*DataFrame*) – Book corpus. It should, at least, have the following columns (with the very same names):
  - *gr_book_id*: book unique identifier.
  - *title*: book titles.
  - *authors*: book authors.
  - *overview*: book overview.
- **vectors_cache_path** (*str, optional*) – Filepath to store the computed vectors or load the vectors from. Defaults to None.
- **encoding_strategy** (*str, optional*) – Encoding strategy to use. The encoding strategy must be a string containing the names of the features to include into the input of the encoder, each of them separated by an underscore ('_'). For example, if you were to use the title and the overview as the encoding strategy, *encoding_strategy* must be either *title_overview* or *overview_title*. Defaults to 'title_overview'.
- **pretrained_biencoder** (*str, optional*) – The model *id* of a predefined *SentenceTransformer* hosted inside a model repo on sbert.net. Defaults to 'paraphrase-distilroberta-base-v2'. Defaults to None.
- **embeddings_cache_path** (*str, optional*) – Filepath to store the computed embeddings or load the embeddings from. Defaults to None.

- **biencoder_max_seq_length** (*int, optional*) – Property to get the maximal input sequence length for the model. Longer inputs will be truncated. Defaults to 512.
- **pretrained_crossencoder** (*str, optional*) – Any model name from Huggingface Models Repository that can be loaded with AutoModel. Defaults to None.

property biencoder
Getter method for *biencoder*.

Returns Current pretrained Bi-Encoder. If none, it attempts to obtain the default one.

Return type SentenceTransformer

property crossencoder
Getter method for *crossencoder*.

Returns Current pretrained Cross-Encoder. If none, it attempts to obtain the default one.

Return type CrossEncoder

search(*query: str*, *k=5*, *k_lexical=20*, *reranking_strategy: Literal[crossencoder, biencoder] = None*)
Perform TF-IDF lexical search.

Parameters

- **query** (*str*) – Textual query.
- **k** (*int, optional*) – Number of most relevant documents to retrieve.
- **k_lexical** (*int, optional*) – If using retrieve and re-rank, number of documents to retrieve by lexical search and re-ranked by any re-ranking strategy. Re-Ranker will return the *k* most relevant entries. *k_lexical* must be greater or equal to *k*. Defaults to 20.
- **reranking_strategy** (*Literal['crossencoder', 'biencoder'], optional*) – Re-ranking strategy to use. Defaults to None

Raises **ValueError** – Raise *ValueError* if *reranking_strategy* takes an ilegal value.

search_multiple(*queries: list*, *write_to: str = None*, ***search_options*)
Perform lexical search for several queries. Refer to the documentation of *search* method for further information.

Parameters

- **queries** (*list*) – Collection of textual queries.
- **write_to** (*str, optional*) – Path of the file in which the results of the query will be written. If the file already exists, existing data will be overwritten. Defaults to None.

setup_biencoder(*pretrained_biencoder='paraphrase-distilroberta-base-v2'*, *max_seq_length=512*)
Set or update the Bi-Encoder to be used to re-rank the results retrieved by TF-IDF search.

Parameters

- **pretrained_model** (*str, optional*) – The model *id* of a predefined *SentenceTransformer* hosted inside a model repo on sbert.net. Defaults to 'paraphrase-distilroberta-base-v2'. Defaults to 'paraphrase-distilroberta-base-v2'.
- **max_seq_length** (*int, optional*) – Property to get the maximal input sequence length for the model. Longer inputs will be truncated. Defaults to 512.

setup_crossencoder(*pretrained_crossencoder='cross-encoder/stsb-distilroberta-base'*)
Set or update the Cross-Encoder to be used to re-rank the results retrieved by TF-IDF search. We recomend using either 'cross-encoder/stsb-distilroberta-base' or any pretrained Cross-Encoder trained on MS MARCO dataset.

Parameters **pretrained_crossencoder** (*str, optional*) – Any model name from Huggingface Models Repository that can be loaded with AutoModel. Defaults to 'cross-encoder/stsb-distilroberta-base'

CHAPTER

THREE

CODE UTILITIES

Documentation for the *code_utils* Python package.

3.1 Utils (`utils.py`)

Set of miscellaneous utilities used across the implementation.

Author: David Lorenzo Alfaro.

`code_utils.utils.`**`append_overviews_to_data`**(*df_overviews: pandas.core.frame.DataFrame*, *df_data: pandas.core.frame.DataFrame*, *overview_index: str = 'gr_book_id'*, *data_index: str = 'gr_book_id'*, *merge_option='right'*)

Merge *df_overviews* with *df_data* using column identifiers *overview_index* and *data_index*, respectively. We allow books with no overviews, hence *right join* is the most suitable operation.

Parameters

- **`df_overviews`** (*`pd.DataFrame`*) – Book overviews.
- **`df_data`** (*`pd.DataFrame`*) – Book data (e.g., title, authors, etc.)
- **`overview_index`** (*`str, optional`*) – Column or index level names to join on in the left DataFrame, defaults to 'gr_book_id'.
- **`data_index`** (*`str, optional`*) – Column or index level names to join on in the right DataFrame, defaults to 'gr_book_id'.
- **`merge_option`** (*`str, optional`*) – Type of merge operation, to be performed can be one of {'left', 'right', 'outer', 'inner'}, to 'right'.

Returns A DataFrame of the two merged objects.

Return type pd.DataFrame

`code_utils.utils.`**`clean_book_title`**(*title: str*, *remove_quotation_marks=True*, *remove_saga_info=False*, *remove_saga_number=True*)

Applies several transformations to a book title to remove noisy data that can potentially affect the performance of the embedding strategies.

Parameters

- **`title`** (*`str`*) – Book title in plain text.
- **`remove_quotation_marks`** (*`bool, optional`*) – If set to True, attempts to remove the quotation marks enclosing the book title, to True.

- **remove_saga_info** (*bool, optional*) – If set to True, attempts to remove information concerning the book saga, defaults to False.
- **remove_saga_number** (*bool, optional*) – If set to True, attempts to remove the saga number, defaults to True.

Returns Processed book title.

Return type str

`code_utils.utils.`**`clean_overview`**`(`*overview: str*`)`

Applies several transformations to a book overview to remove noisy data that can potentially affect the performance of the embedding strategies. One must be careful when applying transformations to the whole corpus because the odds for negative side-effects are high. Here, we attempt to solve some of the problems spotted that, in our tests, should not have any noticeable negative effect on any book overview.

Parameters **overview** (*str*) – Book overview in plain text.

Returns Processed book overview.

Return type str

`code_utils.utils.`**`compute_avg_wordpiece_tokens`**`(`*corpus: list*, *tokenizer=None*`)`

Compute the average number of WordPiece tokens in a list of documents, *corpus*.

Parameters

- **corpus** (*list or array-like*) – List of textual documents.
- **tokenizer** (*BertTokenizer, optional*) – Instance of *BertTokenizer* class. If *None*, it loads the predefined tokenizer of 'bert-base-uncased'.

Returns Average number of WordPiece tokens of the documents in *corpus*.

Return type int

`code_utils.utils.`**`fix_punctuation`**`(`*overview: str*`)`

Attempts to fix some of the identified punctuation issues present in the book overviews.

It is a common issue to find overviews with the following punctuation flaw:

- "[. . .] word.Word [. . .]"
- "[. . .] word!Word [. . .]"
- "[. . .] word?Word [. . .]"

That is to say, spacing after periods, exclamation and question marks is not correctly applied. This lead to some issues when splitting the text into sentences, specially using the NLTK library. Furthermore, it may have other adverse effects on the embedding process (e.g., due to faulty tokenization).

Parameters **overview** (*str*) – Book overview in plain text.

Returns str

Return type Book overview without the identified punctuation flaws.

`code_utils.utils.`**`generate_dataframe_from_sparse_txts`**`(`*base_dir*, *path_standard_format=False*, *out_filename=None*`)`

Generates a dataframe from all *txt* files located in *base_dir*. The dataframe features two columns: *gr_book_id*, an identifier that is retrieved from the name of each *txt* file, and *overview*, containing all information included in the *txt* file.

Parameters

- **base_dir** (*str*) – Directory in which the *txt* files for the book overviews are located.

- **path_standard_format** (*bool, optional*) – Indicates whether the path follows the standard format (backslash separator) or the slash separator, defaults to False.
- **out_filename** (*str, optional*) – Path for the output file, defaults to None.

Returns Returns a dataframe from all txt files located in *base_dir*.

Return type pd.DataFrame

`code_utils.utils.`**`get_bert_model`**(*transformer='bert-base-uncased'*)

Get an instance of the class *BertModel* for the transformer *trasformer*.

Wrapper function of the HuggingFace Transformer's *BertModel* function: *from_pretrained*.

Parameters **transformer** (*str, optional*) – The model *id* of a predefined tokenizer hosted inside a model repo on huggingface.co. Defaults to 'bert-base-uncased'

:return Instance of *BertTokenizer* class. :rtype: BertTokenizer

`code_utils.utils.`**`get_bert_tokenizer`**(*transformer='bert-base-uncased'*)

Get an instance of the class *BertTokenizer* for the transformer *trasformer*.

Wrapper function of the HuggingFace Transformer's *BertTokenizer* function: *from_pretrained*.

Parameters **transformer** (*str, optional*) – The model *id* of a predefined tokenizer hosted inside a model repo on huggingface.co. Defaults to 'bert-base-uncased'

:return Instance of *BertTokenizer* class. :rtype: BertTokenizer

`code_utils.utils.`**`get_crossencoder`**(*crossencoder='cross-encoder/stsb-distilroberta-base', **kwargs*)

Wrapper function to get a *CrossEncoder*.

Parameters **crossencoder** (*str, optional*) – Any model name from Huggingface Models repository that can be loaded with AutoModel. Defaults to 'cross-encoder/stsb-distilroberta-base'.

Returns a CrossEncoder that takes exactly two sentences/texts as input and predicts a score for this sentence pair.It can for example predict the similarity of the sentence pair on a scale of 0 … 1.

Return type CrossEncoder

`code_utils.utils.`**`get_dir_files_content`**(*directory: str, file_extension='.txt'*)

Get the list of files in *directory* with extension *file_extension*.

Parameters

- **directory** (*str*) – Directory in which the files are located. Recursive search is not allowed.
- **file_extension** (*str, optional*) – File extension, defaults to '.txt'

Returns List of tuples (filename, file content).

Return type list

`code_utils.utils.`**`get_file_id_and_content`**(*filepath*)

Returns all textual content in` filepath`

Parameters **filepath** – Path of the text file to read.

Returns Content of the file.

Return type str

`code_utils.utils.`**`get_sentence_transformer`**(*transformer='paraphrase-distilroberta-base-v2'*, *max_seq_length=None*, ***kwargs*)

Wrapper function to get a *SentenceTransformer*.

Parameters

- **`transformer`** (*`str, optional`*) – The model *id* of a predefined *SentenceTransformer* hosted inside a model repo on sbert.net. Defaults to 'paraphrase-distilroberta-base-v2'.
- **`max_seq_length`** (*`int`*) – Property to get the maximal input sequence length for the model. Longer inputs will be truncated. Defaults to None.

Returns a SentenceTransformer model that can be used to map sentences / text to embeddings.

Return type SentenceTransformer

`code_utils.utils.`**`get_tokenized_text`**(*text: str*, *tokenizer=None*)

Get the list of WordPiece tokens in *text*

Parameters

- **`text`** (*`str`*) – Text to tokenize
- **`tokenizer`** (*`BertTokenizer, optional`*) – Instance of *BertTokenizer* class. If *None*, it loads the predefined tokenizer of 'bert-base-uncased'.

Returns list of WordPiece tokens.

Return type list

`code_utils.utils.`**`get_top_k_sentences`**(*document: str*, *k=5*, *embedder=None*, *spacy_model=None*, *preserve_order=True*)

Get the top *k* most meaningful sentences of *document*.

Parameters

- **`document`** (*`str`*) – a document in plain text
- **`k`** (*`int, optional`*) – Top k sentences to return, defaults to 5
- **`embedder`** (*`SentenceTransformer, optional`*) – Instance of *SentenceTransformer*. If *None*, it loads the default pretrained sentence transformer model. Defaults to None.
- **`spacy_model`** (*`str, optional`*) – Name of a spacy pretrained tokenizer model. If *None*, it loads the default model. Defaults to None.
- **`preserve_order`** (*`bool, optional`*) – Preserve the order of the top k sentences with respect to the original document to conserve spatial dependencies between sentences. Defaults to True.

Returns Top *k* most meaningful sentences of *document*.

Return type list

`code_utils.utils.`**`load_corpus`**(*path_corpus: str*, *sep=','*, ***kwargs*)

Wrapper method of Pandas *read_csv* function to load book corpus.

Parameters

- **`path_corpus`** (*`str`*) – Filepath to the corpus.
- **`sep`** (*`str, optional`*) – Delimiter to use, defaults to ','

Returns Corpus DataFrame

Return type DataFrame

`code_utils.utils.`**`load_embeddings`**(*filename: str*, *return_dict_values=True*)

Utility to load embeddings (and other optional stored values) from disk using *pickle*.

Parameters

- **`filename`** (`str`) – Filename of the file to be loaded.
- **`return_dict_values`** (`bool, optional`) – If set to True, returns the values just the values of the dictionary containing all stored data, defaults to True.

Returns Loaded data

`code_utils.utils.`**`makedir`**(*path: str*, *remove_filename=False*, *recursive=True*, *exist_ok=True*)

Creates directory from path if not exists.

Parameters

- **`path`** (`str`) – Path of the directory to be created.
- **`remove_filename`** (`bool, optional`) – If set to True, it attempts to remove the filename from the path, defaults to False
- **`recursive`** (`bool, optional`) – Creates directories recursively (i.e., create necessary subdirectories if necessary), defaults to True
- **`exist_ok`** (`bool, optional`) – is set to False, arises an error if *path* directory exists, defaults to True

`code_utils.utils.`**`prepare_input_encoder`**(*encoding_strategy: str*, *corpus: pandas.core.frame.DataFrame*, *return_input_encoder=True*)

Formats the input to the encoder using the features indicated in *encoding_strategy*. If *encoding_strategy* takes a wrong value this method is likely to fail. Current supported features are 'title', 'authors' and 'overview'. :param str encoding_strategy: The encoding strategy must be a string containing the names of the features to include into the input of the encoder, each of them separated by an underscore ('_'). For example, if you were to use the title and the overview as the encoding strategy, *encoding_strategy* must be either *title_overview* or *overview_title*. :param str path_df: Path in which the dataframe is located. :param return_input_encoder: Return just the collection of inputs to the encoder, defaults to True :type return_input_encoder: bool, optional :return: If *return_input_encoder*, returns the collection of inputs to the encoder. Otherwise, it returns Dataframe including a new column *input_encoder* with the format indicated in *encoding_strategy* :rtype: Dataframe

`code_utils.utils.`**`remove_filename_from_path`**(*out_filename: str*, *path_standard_format=False*)

Attempts to remove filename from the provided path.

Parameters

- **`out_filename`** (`str`) – Filepath.
- **`path_standard_format`** (`bool, optional`) – Indicates whether the path follows the standard format (backslash separator) or the slash separator, defaults to False.

Returns The directory excluding the filename.

Return type str

`code_utils.utils.`**`split_text_into_sentences_nltk`**(*text: str*)

Splits text into sentences using the NLTK library.

Parameters **`text`** (`str`) – Text to be splitted into sentences.

Returns List of sentences in *text*.

Return type list

`code_utils.utils.`**`split_text_into_sentences_spacy`**(*text*, *spacy_model='en_core_web_sm'*)

Splits text into sentences using the Spacy library. SpaCy builds a syntactic tree for each sentence, a robust method that yields more statistical information about the text than NLTK. It performs substancially better than NLTK when using not polished text.

Parameters

- **`text`** (*`str`*) – Text to be splitted into sentences.
- **`spacy_model`** (*`str, optional`*) – Name of the spacy pretrained model used to split text into sentences, defaults to 'en_core_web_sm'.

Returns List of sentences in *text*.

Return type list

`code_utils.utils.`**`store_embeddings`**(*corpus_embeddings*, *out_filename='embeddings.pkl'*, *protocol=5*, ***kwargs*)

Utility to dump embeddings (and other optional values indicated in the keyword arguments) to disk using *pickle*.

Parameters

- **`corpus_embeddings`** – Tensor type data structure containing the embeddings for the corpus.
- **`out_filename`** (*`str, optional`*) – Path for the output file, defaults to 'embeddings.pkl'.
- **`protocol`** – Protocol used for *pickle*, defaults to *pickle.HIGHEST_PROTOCOL*.

`code_utils.utils.`**`summarize_corpus_overviews`**(*corpus_overviews: list*, *top_k=5*, *embedder=None*, *spacy_model=None*, ***kwargs*)

Apply unsupervised Text Summarization techniques to obtain representations for the most meaningful sentences for each document in *corpus_overviews*.

Parameters

- **`corpus_overviews`** (*`list or array-like`*) – Book overviews.
- **`top_k`** (*`int, optional`*) – Number of sentences that will have each overview. Defaults to 5.
- **`embedder`** (*`SentenceTransformer, optional`*) – Instance of *SentenceTransformer*. If *None*, it loads the default pretrained sentence transformer model. Defaults to None.
- **`spacy_model`** (*`str, optional`*) – Name of a spacy pretrained tokenizer model. If *None*, it loads the default model. Defaults to None.

Returns Summarized overviews with at most *top_k* sentences.

Return type list

`code_utils.utils.`**`topk_cos_sim`**(*query_embedding: torch.Tensor*, *embeddings: torch.Tensor*, *top_k: int*)

Get the indices and the cosine similarity score of the *top_k* most similar embeddings to *query_embedding* in *embeddings*.

Parameters

- **`query_embedding`** (*`torch.Tensor`*) – Query embedding.
- **`embeddings`** (*`torch.Tensor`*) – Corpus embeddings.
- **`top_k`** (*`int`*) – Top k most similar to retrieve according to cosine similarity score.

Returns List of scores and list of indexes of the top k results.

Return type (list, list)

3.2 Plotter (`plotter.py`)

Set of plotting utilities used across the implementation.

Author: David Lorenzo Alfaro.

`code_utils.plotter.`**`apply_dimensionality_reduction`**(*word_embeddings, n_components: int, reduction_strategy='pca', random_state=None, _tsne_perplexity=5, _tsne_learning_rate=10, _tsne_n_iter=3000*)

Apply dimensionality reduction on words embeddings using reduction techniques like the Principal Component Analysis (PCA) and the T-distributed Stochastic Neighbour Embedding (t-SNE). The default values for the perplexity, learning rate and number of iterations have been empirically tuned to those that produced acceptable results consistently.

Parameters

- **`word_embeddings`** (*`list or array-like`*) – Collection of word embeddings.
- **`n_components`** (*`int`*) – Dimension of the new embedded space (e.g., 2 for 2D visualization, 3 for 3D visualization).
- **`random_state`** (*`int, optional`*) – Random state for dimensionality reduction techniques, defaults to None
- **`reduction_strategy`** (*`str, optional`*) – Reduction strategy to choose. Can be either 'pca' or 'tsne'. Defaults to 'pca'
- **`_tsne_perplexity`** (*`int, optional`*) – The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Defaults to 5.
- **`_tsne_learning_rate`** (*`int, optional`*) – The learning rate for t-SNE is usually in the range [10.0, 1000.0]. Defaults to 10.
- **`_tsne_n_iter`** (*`int, optional`*) – Maximum number of iterations without progress to abort optimization process, defaults to 3000.

Raises **`ValueError`** – Raise *ValueError* if *reduction_stragy* takes an ilegal value.

Returns Data in the new embedded space.

Return type NumPy array

`code_utils.plotter.`**`display_embeddings_scatterplot_2D`**(*word_embeddings, color_text=None, random_state=None, reduction_strategy='pca', _graph_showlegend=True, tsne_params={}, **scatter_params*)

Visualize BERT embeddings in a 2D scatterplot using dimensionality reduction techniques like Principal Component Analysis (PCA) and the T-distributed Stochastic Neighbour Embedding (t-SNE).

Parameters

- **word_embeddings** (*list or array-like*) – Collection of word embeddings.
- **color_text** (*str, optional*) – Label for plotly scatterplot *color* attribute. Defaults to None
- **random_state** (*int, optional*) – Random state for dimensionality reduction techniques, defaults to None
- **reduction_strategy** (*str, optional*) – Reduction strategy to choose. Can be either 'pca' or 'tsne'. Defaults to 'pca'
- **_graph_showlegend** (*bool, optional*) – Show legend, defaults to True
- **tsne_params** – Additional parameters for t-SNE, defaults to {}

code_utils.plotter.**display_embeddings_scatterplot_3D**(*word_embeddings, color_text: str = None, random_state=None, reduction_strategy='pca', _graph_showlegend=True, tsne_params={}, **scatter_params*)

Visualize BERT embeddings in a 3D scatterplot using dimensionality reduction techniques like Principal Component Analysis (PCA) and the T-distributed Stochastic Neighbour Embedding (t-SNE).

Parameters

- **word_embeddings** (*list or array-like*) – Collection of word embeddings.
- **color_text** (*str, optional*) – Label for plotly scatterplot *color* attribute. Defaults to None
- **random_state** (*int, optional*) – Random state for dimensionality reduction techniques, defaults to None
- **reduction_strategy** (*str, optional*) – Reduction strategy to choose. Can be either 'pca' or 'tsne'. Defaults to 'pca'
- **_graph_showlegend** (*bool, optional*) – Show legend, defaults to True
- **tsne_params** – Additional parameters for t-SNE, defaults to {}

code_utils.plotter.**histogram_embeddings_nn**(*data: list*)

Plot histogram for the nearest neighbors of an embedding.

Parameters **data** (*list or array-like*) – Two dimensional array containing a collection of similar words (first component), list of similarity scores (second component), and a list of labels (third component).

code_utils.plotter.**plot_bert_embeddings_nn**(*model, vocab: dict, input_words, k=5, display_option='3d', reduction_strategy='pca', random_state=None*)

Visualize the *k* most similar words in *vocab* in the 2D or 3D embedding space. (Disclaimer: sorry about poor code readability).

Parameters

- **model** (*BertModel*) – Bert pretrained model.
- **vocab** (*dict*) – Tokenizer vocabulary.
- **input_words** (*Iterable*) – Words, the KNN of which are to be calculated and displayed.
- **k** (*int, optional*) – number of similar words to visualize. By default, 5

- **display_option** (*str, optional*) – Visualize BERT embeddings either in '2d' or '3d', defaults to '3d'
- **reduction_strategy** (*str, optional*) – Reduction strategy to choose. Can be either 'pca' or 'tsne'. Defaults to 'pca'
- **random_state** (*int, optional*) – Random state for dimensionality reduction techniques, defaults to None

Raises **ValueError** – Raise *ValueError* if *display_option* takes an ilegal value.

code_utils.plotter.**plot_heatmap**(*data*, *color_continuous_scale: str = None*, ***kwargs*)

Plot heatmap. Wrapper of the plotly *imshow* function.

Parameters

- **data** (*array-like*) – 2D data to be plotted.
- **color_continuous_scale** (*str, optional*) – Colour scale to be used in the heatmap, defaults to None

code_utils.plotter.**plot_heatmap_embeddings**(*model*, *tokenizer*, *data: pandas.core.frame.DataFrame*, *polysemous_word: str*, *color_continuous_scale: str = None*)

Plot heatmap of the cosine similarity of all different contextual embeddings of *polysemous_word* in *data*. This experiment is explained in the "Exploring BERT contextual representations" section of the dissertation. (Disclaimer: sorry about poor code readability).

Parameters

- **model** (*BertModel*) – Bert pretrained model.
- **tokenizer** (*BertTokenizer*) – Bert precomputed tokenizer.
- **data** (*pd.DataFrame*) – Test data for WSD evaluation.
- **polysemous_word** (*str*) – Polysemous word.
- **color_continuous_scale** (*str, optional*) – Colour scale to be used in the heatmap, defaults to None

Returns The collection of the different contextual embeddings, along with the labels.

code_utils.plotter.**plot_scatter_with_secondary_y_axis**(*x*, *y*, *y2*, *fig_title=''*, *x_title=''*, *y_title=''*, *y2_title=''*)

Plot scatter plot with secondary y axis.

code_utils.plotter.**write_embeddings_to_disk**(*model*, *vocab: dict*, *out_dir='runs/bert_embeddings'*, *write_word_embeddings=True*, *write_position_embeddings=False*, *write_type_embeddings=False*)

Utility to write embeddings weights to disk that can be loaded with TensorBoard to visualize the embeddings.

Parameters

- **model** (*BertModel*) – Bert pretrained model.
- **vocab** (*dict*) – Tokenizer vocabulary.
- **out_dir** (*str, optional*) – Directory to write the embeddings, defaults to 'runs/bert_embeddings'.

- **write_word_embeddings** (*bool, optional*) – Write word embeddings to disk, defaults to True.
- **write_position_embeddings** (*bool, optional*) – Write position embeddings to disk, defaults to False.
- **write_type_embeddings** (*bool, optional*) – Write token type embeddings to disk, defaults to False.

CHAPTER

FOUR

CODE USED FOR EXPERIMENTS.

Documentation for the experiments described in the dissertation.

4.1 Speedup-recall tradeoff depending on the number of trees used in ANNOY(`experiment_annoy_ntrees.py`)

Experiment to test peedup-recall tradeoff depending on the number of trees used in ANNOY.

Author: David Lorenzo Alfaro.

`experiment_annoy_ntrees.`**`evaluate_n_trees`**(*queries: list*, *search_alg*, *k=5*, *verbose=True*)

Utility used to evaluate the speedup-recall tradeoff of ANNOY as the number of trees increases.

Parameters

- **`queries`** (*`list or array-like`*) – Collection of textual queries.
- **`search_alg`** (`SemanticSearch`) – Semantic search object.
- **`k`** (*`int, optional`*) – Top k elements to consider in the comparison, defaults to 5
- **`verbose`** (*`bool, optional`*) – Verbosity mode, defaults to True

4.2 Evaluate text summarization using different values of *k* top sentences (`experiment_text_summarization.py`)

Experiment to evaluate text summarization using different values of *k* top sentences.

Author: David Lorenzo Alfaro.

`experiment_text_summarization.`**`evaluate_summarization_candidates`**(*corpus_overviews: list*, *candidates*, *embedder=None*, *spacy_model=None*, *tokenizer=None*, ***kwargs*)

Get an array of reduction rates and number of word pieces for a set of candidate number of sentences used to summarize each book overview (must be a list or array-like of integers).

Parameters

- **corpus_overviews** (*list or array-like*) – Book overviews.
- **candidates** (*Iterable*) – List of number of candidates to evaluate.
- **embedder** (*SentenceTransformer, optional*) – Instance of *SentenceTransformer*. If *None*, it loads the default pretrained sentence transformer model. Defaults to None.
- **spacy_model** (*str, optional*) – Name of a spacy pretrained tokenizer model. If *None*, it loads the default model. Defaults to None.
- **tokenizer** (*BertTokenizer, optional*) – Instance of *BertTokenizer* class. If *None*, it loads the predefined tokenizer of 'bert-base-uncased'.

Returns Summarized overviews with at most *candidates* sentences.

Return type list

CHAPTER

FIVE

VISUALIZATION OF BERT EMBEDDINGS.

As part of the undergraduate dissertation: Similarity Measures in Natural Language Processing based on Deep Learning Models.

David Lorenzo Alfaro

5.1 Introduction.

BERT embeddings have a dimensionality of 768. As studied, linguistic features are be encoded along the representation of the word in a distributed fashion. Whilst embeddings work particularly well for computer systems to handle semantic and syntactic information about natural language, if humans were to visualize sequences of 768 floating point values, observing any valuable information would be an extremely difficult task. Furthermore, when handling natural language, the most common scenario involves dealing with sequences of words, thus posing neural NLP as a high-dimensionality problem. Consequently, the great majority of visualization techniques for embeddings involves dimensionality reduction.

The most common dimensionality reduction techniques are the Principal Component Analysis (PCA) and the T-distributed Stochastic Neighbour Embedding (t-SNE). The former is usually preferred because it can be used as a black box. That is, since it is a non-parametric method, it is not usually required to know the mathematics behind PCA.

The following sections provide a general overview of the different experiments carried out to visualize BERT embeddings. To that end, the increasingly popular open source graphing library Plotly (https://plotly.com/) is used. Plotly is a simple but expressive capable library that enables creating interactive charts and maps, a pretty much desired characteristic for the visualization tasks hereinafter described.

5.2 Visualization of kNN neighbours of BERT's pretrained embeddings.

The first test consist in gathering the WordPiece token embeddings weights learned through training of a BERT model for some query words to then find the k nearest neighbours to each word in the query. Let us first load all necessary libraries and modules.

[1]:
```
import numpy as np
import pandas as pd
from code_utils.paths import *
# Plotter module contain a bunch of utilities for
# BERT embeddings visualization.
from code_utils import utils, plotter
```

For the sake of simplicity, we decided to use BERT's base uncased pretrained model. This model has way less parameters and weights, hence is lighter. We have not run this experiment on any pretrained model based on BERT Large architecture, albeit differences in outcomes, if any, must be negligible.

[2]:
```
pretrained_model = 'bert-base-uncased'

# Get BERT pretained model.
model = utils.get_bert_model(pretrained_model)

# Get BERT precomputed tokenizer.
tokenizer = utils.get_bert_tokenizer(pretrained_model)

# Get tokenizer vocabulary.
vocab = tokenizer.vocab

# To guarantee experiment reproducibility
random_state = 0
```

It is worth mentioning that the way in which experiment has been run is not appropriate for BERT for several reasons. First and foremost, BERT derives context-sensitive subword representations. For BERT to actually derive the proper embedding for a specific token, some context (i.e., a sentence containing that word) must be provided. In the approach we are following, comparisons among representations make no use of context, which would just be fine for context-independent embeddings such as those derived by techniques like GloVe or Word2vec. Second, since no actual embeddings are being computed (i.e., the learned weights for each token in BERT's vocab are used), all query words do necessarily need to belong to the BERT's model vocab, hence noticeably constraining the range of possible words to use in the query. Under these assumptions, we consider this experiment, albeit not entirely correct, can help the reader to have a better understanding of some of the concepts previously discussed, especially those concerning the linguistic regularities in continuous space word representations.

The process is straightforward. Given a subset of BERT's vocabulary tokens, get the learned embeddings and compute the k nearest neighbours to each query word. The set of neighbour candidates is the complete vocabulary used in BERT (30,000 tokens). Since it is a sine qua non condition that the query tokens belong to BERT's vocab, the nearest neighbour for each query token is necessarily the token itself. We found that the easiest and computationally cheapest way to prevent this from happening is to get the $k+1$ nearest neighbours to then discard those that are incorrect.

Afterwards, the proper embeddings for the kNN of all query tokens are fed into one dimensionality reduction technique. Since it is our will that the output of either PCA or t-SNE are visually interpretable, the embeddings, originally in the 768-dimensional space, are compressed into either 2-dimensional or 3-dimensional representations, attempting to preserve some of the meaningful information.

For further details, check the documentation for `plot_bert_embeddings_nn` method.

In the dissertation we discussed that Word2vec embeddings were able to capture features like the notion of gender or the syntactic singular/plural relation. For this first query, we will be using again the word queen.

[3]:
```
query = 'queen'

# Format query words.
input_words = query.replace(' ','').lower().split(',')

k=8

# Invoke method to plot PCA 2D projection of the embeddings
# and a histogram for all nearest neighbours of the query
# word embeddings.
plotter.plot_bert_embeddings_nn(model,
```

(continues on next page)

(continued from previous page)

```
                                vocab,
                                input_words,
                                k=k,
                                reduction_strategy='pca',
                                display_option='2d',
                                random_state=random_state)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

As it can be seen, the eight closest words to queen are, in order, king, queens, princess, empress, prince, duchess, countess and monarch. The results are astoundingly positive: all neighbours are reasonably related, either syntactically (e.g., queens) or semantically (e.g., king). Furthermore, the similarity scores obtained denote strong relativeness between the similar and query words.

It is also worth noting that, excepting king and prince (which, on their own are fairly related to queen), all remaining nearest neighbours are royal titles held by women, which hints BERT embeddings capability to capture the notion of gender. Moreover, the words empress, countess, and duchess seem to be closer in space, probably due to syntactic similarities among them (i.e., they all are suffixed with "ess").

Let us now repeat the experiment using a set of queries

[4]:
```
queries = 'stupid, queen, wizard, spain, brother'

# Format query words.
input_words = queries.replace(' ','').lower().split(',')

k=5

# Invoke method to plot T-SN 2D projection of the embeddings
# and a histogram for all nearest neighbours of the query
# word embeddings.
plotter.plot_bert_embeddings_nn(model,
                                vocab,
                                input_words,
                                k=k,
                                reduction_strategy='t-sne',
                                display_option='2d',
                                random_state=random_state)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

As hinted by the results, each query word, along with its most similar words in terms of cosine-similarity, conform a cluster. Unless query words are highly correlated, this is to expect, being as cosine-similarity is a measurement of distance in the d-dimensional space. Consistently with the highlighted observations for the previous experiment, all nearest neighbours to the query words are intimately connected syntactically and/or semantically, with the cosine similarity scores indicating strong relatedness.

Generally, some of the words populating the nearest neighbour set of each query word are synonyms (e.g., wizard and magician, stupid and dumb). Also, for the query word Spain, some of the most similar words are countries as well (e.g., Portugal, Italy). BERT embeddings showcase, once more, its capability to learn fine-grained features about natural language. For the query word brother some of the similar words are terms that belong to the lineal kinship system used in the English-speaking world (e.g., son, father).

It is also worth noting that, according to empirical observations, t-SNE dimensionality reduction technique is encouraged over PCA as the number of query words increase, being as it has reportedly produced better defined clusters, which favours the proper visualization of the results. Unlike PCA, t-SNE is a parametric method. The default values for the perplexity, learning rate and number of iterations have been empirically tuned to those that produced acceptable results consistently.

5.2.1 Visualizing BERT embeddings with Tensorboard.

Yet another manner to visualize BERT is using TensorBoard , the TensorFlow's open source visualization toolkit which provides all the necessary logic to project the embeddings to a lower dimensional space and to make queries in real time. To that end, we have to write the word embeddings learned weights into disk using a utility described in the `plotter` python module.

[5]:
```
# Invoking a method that writes embeddings weights to disk that
# can be loaded with TensorBoard to visualize the embeddings.
out_dir = 'runs/bert_embeddings'
plotter.write_embeddings_to_disk(model,
                                 vocab,
                                 out_dir=out_dir,
                                 write_word_embeddings=True,
                                 write_position_embeddings=False,
                                 write_type_embeddings=False)
```

You can then run TensorBoard. For example, for `out_dir = 'runs/bert_embeddings`, you would need to input the following command on a Python console:

```
tensorboard --logdir="<current notebook folder path>\runs"
```

5.3 Exploring BERT contextual representations.

As previously studied, the power of BERT lies in its ability to derive representations based on context. In this experiment, the embeddings of a word in different contexts (i.e., in different sentences) were computed to check whether there are significant differences among them.

To that end, we used a public domain licensed dataset for word sense disambiguation (WSD) available in Kaggle. The dataset file is an excel file with three columns. The first is the serial number (SN), which assigns a unique identifier to each tuple of contextual sentence (second column) and target polysemous word contained in the sentence (third column).

[6]:
```
PATH_WSD_DATASET = 'visualization/test data for WSD evaluation _2905.xlsx'

# Load WSD dataset.
df_polysemy = pd.read_excel(PATH_WSD_DATASET)
```

(continues on next page)

(continued from previous page)

```
# Set the serial number column as the index column.
df_polysemy = df_polysemy.set_index(df_polysemy.sn)

# Let us sample ten random instances.
df_polysemy.sample(n=10, random_state=random_state)
```

[6]:

```
        sn                                   sentence/context polysemy_word
sn
583    583                                 I lost my AC remote.        remote
1812  1812                                a young man is running           man
2250  2250                          Crane is a large water bird.         crane
1653  1653                         Insert the jack in the LAN port         jack
668    668                         Thank you for your prompt reply.       prompt
515    515                             She likes Russian pop song.           pop
619    619  Films are rated on a scale of poor, fair, good...         scale
2692  2692                      The pilot is checking belly of plane         belly
2491  2491  We want to appeal to our core supporters witho...          core
937    937  Any type of single cut metal file can be used ...          file
```

Let us now use one of the many polysemous words in the dataset.

[7]:

```
polysemous_word='bank'
df_polysemy[df_polysemy['polysemy_word']== polysemous_word]
```

[7]:

```
    sn                                   sentence/context polysemy_word
sn
1    1                                 I have bank account.          bank
2    2                     Loan amount is approved by the bank.          bank
3    3  He returned to office after he deposited cash ...          bank
4    4     They started using new software in their bank.          bank
5    5                       he went to bank balance inquiry.          bank
6    6  I wonder why some bank have more interest rate...          bank
7    7  You have to deposit certain percentage of your...          bank
8    8                              He took loan from a Bank.          bank
9    9                         he is waking along the river bank.          bank
10  10             The red boat in the bank is already sold.          bank
11  11  Spending time on the bank of Kaligandaki river...          bank
12  12            He was sitting on sea bank with his friend          bank
13  13  She has always dreamed of spending a vacation ...          bank
14  14   Bank of a river is very pleasant place to enjoy.          bank
```

As it can be seen, the dataset contains fourteen different sentences with the target word bank. Sentences from 1 to 8 refer to bank as as *"an organization where people and businesses can invest or borrow money, change it to foreign money, etc., or a building where these services are offered"*; whereas sentences from 9 to 14 refer to bank as a *"sloping raised land, especially along the sides of a river"* (definitions from Cambridge Dictionary). It would be desirable for BERT to be able to disambiguate both senses of the word, which would translate to being capable of deriving distant representations in the n-dimensional space for the different meanings of bank. As we have studied, the self-attention mechanism in BERT's model architecture is responsible for baking into the representation of each word in a sequence salient information about the rest of the words in the sequence.

This outstanding characteristic of the created representations by BERT exhibit an intrinsic degree of natural language understanding never seen any time before (even at pretraining!). For instance, the context words in the first three sentences of the previous table are different and refer to distinct concepts (e.g., account, loan, amount, deposited, cash). That notwithstanding, they are all related because they all belong to a semantic field of interconnected words that can be used in similar contexts.

Let us test whether BERT realizes this connection and embeds it into the resultant representations.

```
[8]: context_embeddings, labels = plotter.plot_heatmap_embeddings(model,
                                                                  tokenizer,
                                                                  df_polysemy,
                                                                  polysemous_word)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

```
Context sentence for each contextual embedding of 'bank'.

bank_1: I have bank account.
bank_2: Loan amount is approved by the bank.
bank_3: He returned to office after he deposited cash in the bank.
bank_4: They started using new software in their bank.
bank_5: he went to bank balance inquiry.
bank_6: I wonder why some bank have more interest rate than others.
bank_7: You have to deposit certain percentage of your salary in the bank.
bank_8: He took loan from a Bank.
bank_9: he is waking along the river bank.
bank_10: The red boat in the bank is already sold.
bank_11: Spending time on the bank of Kaligandaki river was his way of enjoying in␣
↪his childhood.
bank_12: He was sitting on sea bank with his friend
bank_13: She has always dreamed of spending a vacation on a bank of Caribbean sea.
bank_14: Bank of a river is very pleasant place to enjoy.
```

As it can be seen, BERT seems to, in fact, realize of this connection. Therefore, the embeddings for the target word when referred as a financial institution exhibit strong similarity (the contextual baked into the embedding is similar). Analogously, the embeddings for bank in sentences from 9 to 14 are very similar, and they are all farther from those of sentences from 1 to 8, excepting that of the tenth sentence (arguably because the word sold is in the sentence).

```
[9]: plotter.display_embeddings_scatterplot_3D(context_embeddings,
                                               reduction_strategy='tsne',
                                               random_state=random_state,
                                               _graph_showlegend=False,
                                               title='Projection for different contextual␣
↪embeddings',
                                               text=labels,
                                               color=labels)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

CHAPTER

SIX

EXPLORATORY DATA ANALYSIS AND DATA PREPROCESSING.

As part of the undergraduate dissertation: Similarity Measures in Natural Language Processing based on Deep Learning Models.

David Lorenzo Alfaro

6.1 Introduction.

To implement a semantic similarity search information retrieval system, a collection of resources is required. We can, then, use similarity measures between searches and data, willing to retrieve relevant information in an efficient fashion.

In our work, we will be using information about ten thousand books from the goodbooks-10k kaggle dataset. All the information was retrieved from Goodreads, the world's largest site for readers and book recommendations.

The original dataset has been previously modified to better manage the different identifiers and indexes available for each book.

Fortunately, the great majority of word and sentence embedding techniques have been trained on large corpora, often involving humongous number of books (e.g., Toronto Book Corpus). This results on representations that, out of the box, offer great performance for a wide variety of downstream tasks with little fine-tuning being required.

6.2 Dataset exploration.

Before getting started, it is first necessary to load all libraries and dependencies that will be used later in the notebook.

[4]:
```
import os
import pandas as pd
import numpy as np
from code_utils import utils
from code_utils.paths import *
from pathlib import Path
```

Besides, we set a seed to guarantee reproducibility of the experiments.

[5]:
```
seed = 0
```

Let us now unzip containing the collection of books.

[6]:
```
from shutil import unpack_archive
unpack_archive(PATH_DATASET_ZIP, DIR_DATASET)
```

```
[7]: filepath = PATH_BOOKS
     data = pd.read_csv(filepath, sep='	')
```

To sample the first *n* instances of a dataset we can use the `head` function.

```
[8]: data.head(5)
```

```
[8]:    book_id  gr_book_id  gr_best_book_id  work_id  books_count       isbn  \
0        0     2767052          2767052  2792775          272  439023483
1        1           3                3  4640799          491  439554934
2        2       41865            41865  3212258          226  316015849
3        3        2657             2657  3275794          487   61120081
4        4        4671             4671   245494         1356  743273567

         isbn13                      authors  original_publication_year  \
0  9.780439e+12              Suzanne Collins                     2008.0
1  9.780440e+12  J.K. Rowling, Mary GrandPré                     1997.0
2  9.780316e+12              Stephenie Meyer                     2005.0
3  9.780061e+12                   Harper Lee                     1960.0
4  9.780743e+12          F. Scott Fitzgerald                     1925.0

                             original_title  ... ratings_count  \
0                          The Hunger Games  ...       4780653
1  Harry Potter and the Philosopher's Stone  ...       4602479
2                                  Twilight  ...       3866839
3                     To Kill a Mockingbird  ...       3198671
4                          The Great Gatsby  ...       2683664

  work_ratings_count  work_text_reviews_count  ratings_1  ratings_2  \
0            4942365                   155254      66715     127936
1            4800065                    75867      75504     101676
2            3916824                    95009     456191     436802
3            3340896                    72586      60427     117415
4            2773745                    51992      86236     197621

   ratings_3  ratings_4  ratings_5  \
0     560092    1481305    2706317
1     455024    1156318    3011543
2     793319     875073    1355439
3     446835    1001952    1714267
4     606158     936012     947718

                                           image_url  \
0  https://images.gr-assets.com/books/1447303603m...
1  https://images.gr-assets.com/books/1474154022m...
2  https://images.gr-assets.com/books/1361039443m...
3  https://images.gr-assets.com/books/1361975680m...
4  https://images.gr-assets.com/books/1490528560m...

                                     small_image_url
0  https://images.gr-assets.com/books/1447303603s...
1  https://images.gr-assets.com/books/1474154022s...
2  https://images.gr-assets.com/books/1361039443s...
3  https://images.gr-assets.com/books/1361975680s...
4  https://images.gr-assets.com/books/1490528560s...

[5 rows x 23 columns]
```

Alternatively, we can use the `sample` function, which samples *n* random instances of the dataset.

[9]:
```
data.sample(5, random_state=seed)
```

[9]:
```
      book_id  gr_book_id  gr_best_book_id  work_id  books_count        isbn  \
9394     9394       38703            38703   575142           43   385733143
898       898       53835            53835  1959512          836  159308143X
2398     2398       43893            43893  1443364           47   765344300
5906     5906       31244            31244  2888469          378   375761144
2343     2343      497199           497199  1132770           80   876852630

            isbn13                      authors  original_publication_year  \
9394  9.780386e+12                 Louis Sachar                     2006.0
898   9.781593e+12  Edith Wharton, Maureen Howard                     1920.0
2398  9.780765e+12               Terry Goodkind                     2003.0
5906  9.780376e+12              Charles Dickens                     1865.0
2343  9.780877e+12             Charles Bukowski                     1975.0

                          original_title  ... ratings_count work_ratings_count  \
9394                         Small Steps  ...         11837              13095
898                 The Age of Innocence  ...        102646             114994
2398  Naked Empire (Sword of Truth, #8)  ...         39682              42066
5906                    Our Mutual Friend  ...         18599              20659
2343                            Factotum  ...         37376              40444

      work_text_reviews_count  ratings_1  ratings_2  ratings_3  ratings_4  \
9394                     1387        267       1177       4066       4471
898                      5051       2359       6549      25631      42542
2398                      548       1519       3639       9953      12891
5906                     1102        434        986       3803       6936
2343                     1213        457       1875       8979      16585

      ratings_5                                          image_url  \
9394       3114  https://s.gr-assets.com/assets/nophoto/book/11...
898       37913  https://s.gr-assets.com/assets/nophoto/book/11...
2398      14064  https://s.gr-assets.com/assets/nophoto/book/11...
5906       8500  https://images.gr-assets.com/books/1403189244m...
2343      12548  https://images.gr-assets.com/books/1407706616m...

                                          small_image_url
9394  https://s.gr-assets.com/assets/nophoto/book/50...
898   https://s.gr-assets.com/assets/nophoto/book/50...
2398  https://s.gr-assets.com/assets/nophoto/book/50...
5906  https://images.gr-assets.com/books/1403189244s...
2343  https://images.gr-assets.com/books/1407706616s...

[5 rows x 23 columns]
```

As it can be observed, the dataset has 23 different features. However, only the first and last 10 features of the dataset are being displayed. Let us print the names of all the features in the dataset.

[10]:
```
data.columns
```

[10]:
```
Index(['book_id', 'gr_book_id', 'gr_best_book_id', 'work_id', 'books_count',
       'isbn', 'isbn13', 'authors', 'original_publication_year',
       'original_title', 'title', 'language_code', 'average_rating',
       'ratings_count', 'work_ratings_count', 'work_text_reviews_count',
       'ratings_1', 'ratings_2', 'ratings_3', 'ratings_4', 'ratings_5',
       'image_url', 'small_image_url'],
      dtype='object')
```

Here's a brief description for the features in the dataset.

- books_count represents the number of editions for a given work.
- gr_best_book_id contains the most popular edition for a given work.
- Columns book_id, gr_book_id, gr_best_book_id, work_id, isbn and isbn13 are different identifiers for the book. As we will see later, the book overviews are not included in this dataset and have been obtained by means of scraping. Each overview is identified with the gr_book_id identifier, thus it is the link between both sources of information. Let us first check that it is a valid identifier (i.e., there are no null values and all identifiers are unique).

[11]:
```
print(f"Unique values in 'gr_book_id' column: {len(data.gr_book_id.unique())}")
print(f"Print null values in 'gr_book_id' column {data.gr_book_id[data.gr_book_id.
↪isna()]}")
```

```
Unique values in 'gr_book_id' column: 10000
Print null values in 'gr_book_id' column Series([], Name: gr_book_id, dtype: int64)
```

- gr_book_id is, in fact, a valid identifier, thus it is the one that we will be using. Furthermore, we also now know that there are no duplicated instances in the dataset, since at least one of its features has no repeated values. The remaining identifier columns can be deleted, as they do not provide any more meaningful information for the tasks that we are to perform.
- As the name suggests, authors contains the names of the authors of the book.
- original_publication_year indicates the year in which the book was published. We will not be using this information.
- title is the english title of the book.
- original_title is the title of the book in its original language. We are primarily concerned with english textual information, hence title is a more suitable feature.
- language_code indicates the textual code assigned to the language of the book. This feature is particularly useful because it will help us get rid of non-English books.
- average_rating is a floating value indicating the average rating of a book, ranging from 1 to 5. This feature does not provide relevant information for semantic search, thus it will be discarded. That notwithstanding, it could be used as a criteria to filter the query results, prioritizing those that have better ratings.
- ratings_count indicates the number of registered ratings for a book. Analogously, work_ratings_count and work_text_reviews_count indicate the number of ratings and reviews a work has in the platform, respectively. None of this information is useful for our work.
- ratings_1, ratings_2, ratings_3, ratings_4 and ratings_5 characteristics hold the counts for each rating value. Again, this feature does not provide any relevant information to perform semantic search.
- image_url and small_image_url contain links to pictures of the book cover. Since images cannot be displayed in CLIs, we will discard this information too.

6.3 Remove useless features.

Let's get rid of all not useful features.

```
[12]: columns_to_drop = set(['book_id', 'gr_best_book_id', 'work_id', 'books_count',
              'isbn', 'isbn13', 'original_publication_year', 'original_title',
              'average_rating','ratings_count', 'work_ratings_count',
              'work_text_reviews_count', 'ratings_1', 'ratings_2', 'ratings_3',
              'ratings_4', 'ratings_5', 'image_url', 'small_image_url'])
```

```
[13]: data = data.drop(columns_to_drop, axis=1)
      data.sample(5, random_state=seed)
```

```
[13]:       gr_book_id                          authors  \
      9394       38703                     Louis Sachar
      898        53835  Edith Wharton, Maureen Howard
      2398       43893                   Terry Goodkind
      5906       31244                  Charles Dickens
      2343      497199                 Charles Bukowski

                                         title language_code
      9394            Small Steps (Holes, #2)           eng
      898                The Age of Innocence           eng
      2398  Naked Empire (Sword of Truth, #8)         en-GB
      5906                  Our Mutual Friend           eng
      2343                           Factotum           NaN
```

6.4 Integrate book overviews into the dataset.

Now that all useless characteristics have been deleted, let's append the overviews to the dataframe. The book overviews are stored in a directory, one `txt` file for each overview. We will first generate a dataframe containing all txt files in the directory. The filename for each `txt` file is the `gr_book_id` identifier.

```
[14]: book_overviews = utils.generate_dataframe_from_sparse_txts(DIR_OVERVIEW)
```

```
[15]: print(f"Number of overviews in the dataset: {book_overviews.overview.shape[0]}")
      book_overviews
```

```
Number of overviews in the dataset: 9956
```

```
[15]:       gr_book_id                                           overview
      0              1  When Harry Potter and the Half-Blood Prince op...
      1             10  Six years of magic, adventure, and mystery mak...
      2       10000191  À sa naissance, Lisbeth est enlevée à sa mère ...
      3          10006  The discovery of a mysterious notebook turns a...
      4        1000751  When orphaned 11-year-old Pollyanna comes to l...
      ...          ...                                                ...
      9951     9995135  At long last, New York Times bestselling autho...
      9952       99955  Paine's daring prose paved the way for the Dec...
      9953        9998  The Woman in the Dunes, by celebrated writer a...
      9954     9998705  FLASH! Illuminated by lightning, a lifeless hu...
      9955     9999107  Witty, moving, and brilliantly entertaining, T...

      [9956 rows x 2 columns]
```

As it can be seen, there are 9956 book overviews, which is less than the number of instances in the other dataframe. Consequently, at least 44 books will have no overview. There are several strategies to merge both dataframes. In this case, we will allow having books with no overview (*left join* operation).

[16]:
```
data = pd.merge(data, book_overviews, left_on='gr_book_id', right_on='gr_book_id',␣
↪how='left')
data
```

[16]:
```
      gr_book_id                         authors  \
0        2767052                 Suzanne Collins
1              3  J.K. Rowling, Mary GrandPré
2          41865                 Stephenie Meyer
3           2657                      Harper Lee
4           4671            F. Scott Fitzgerald
...          ...                             ...
9995     7130616                  Ilona Andrews
9996      208324                  Robert A. Caro
9997       77431                Patrick O'Brian
9998     8565083                Peggy Orenstein
9999        8914                    John Keegan

                                                  title language_code  \
0                 The Hunger Games (The Hunger Games, #1)           eng
1      Harry Potter and the Sorcerer's Stone (Harry P...           eng
2                                Twilight (Twilight, #1)         en-US
3                                  To Kill a Mockingbird           eng
4                                       The Great Gatsby           eng
...                                                   ...           ...
9995                          Bayou Moon (The Edge, #2)           eng
9996      Means of Ascent (The Years of Lyndon Johnson, #2)           eng
9997                                 The Mauritius Command           eng
9998      Cinderella Ate My Daughter: Dispatches from th...           eng
9999                                   The First World War           NaN

                                               overview
0     Winning will make you famous. Losing means cer...
1     Harry Potter's life is miserable. His parents ...
2     About three things I was absolutely positive.F...
3     The unforgettable novel of a childhood in a sl...
4     On its first publication in 1925, The Great Ga...
...                                                 ...
9995  The Edge lies between worlds, on the border be...
9996  Robert A. Caro's life of Lyndon Johnson, which...
9997  "O'Brian's Aubrey-Maturin volumes actually con...
9998  The acclaimed author of the groundbreaking bes...
9999  The First World War created the modern world. ...

[10000 rows x 5 columns]
```

6.5 Remove instances with invalid language codes.

Let us now check whether there are noisy data in any of the selected characteristics. Starting off with the language code, we need to make sure that all data fed into the models is in English, being as they have been trained to derive semantic representations for English texts. To that end, let's see how many language codes are in the dataset.

[17]:
```
data.language_code.unique()
```

[17]:
```
array(['eng', 'en-US', 'en-CA', nan, 'spa', 'en-GB', 'fre', 'nl', 'ara',
       'por', 'ger', 'nor', 'jpn', 'en', 'vie', 'ind', 'pol', 'tur',
       'dan', 'fil', 'ita', 'per', 'swe', 'rum', 'mul', 'rus'],
      dtype=object)
```

As it can be seen, there are plenty of different languages. However, is the title and the overview of the book written in the language indicated in `language_code`? Let's test it on some of the books with `language_code = spa`

[18]:
```
data[data.language_code == 'spa'].sample(n=10, random_state=seed)
```

[18]:
```
      gr_book_id                                     authors  \
9472       53809                                Paulo Coelho
83          7677                            Michael Crichton
9890     1365225                          José Emilio Pacheco
3751      140302                             Agatha Christie
4508       63032                              Roberto Bolaño
9222       61794                                   Anonymous
3476       31343                                  Anne Rice
5125       53926                          Mario Vargas Llosa
1799       22590  Philip K. Dick, David Alabort, Manuel Espín
555        10603                                Stephen King

                                         title language_code  \
9472                                    Maktub           spa
83          Jurassic Park (Jurassic Park, #1)           spa
9890                 Las batallas en el desierto           spa
3751   Poirot Investiga (Hércules Poirot, #3)           spa
4508                                      2666           spa
9222            La vida del Lazarillo de Tormes           spa
3476  Pandora (New Tales of the Vampires, #1)           spa
5125                Travesuras de la niña mala           spa
1799                                      Ubik           spa
555                                       Cujo           spa

                                               overview
9472  Maktub não é um livro de conselhos, mas uma tr...
83                                                  NaN
9890  Historia de un amor imposible, narración de un...
3751                                                NaN
4508  A cuatro profesores de literatura, Pelletier, ...
9222  Lázaro es un muchacho desarrapado a quien la m...
3476  Anne Rice, creator of the Vampire Lestat, the ...
5125  ¿Cuál es el verdadero rostro del amor?Ricardo ...
1799  Ubik is a 1969 science fict...
555   Outside a peaceful town in central Maine, a mo...
```

Since the title and the overview seems to be written in the language indicated in `language_code`, we will only choose those language codes mapped to English texts: `eng`, `en-US`, `en-CA`, `en-GB` and `en`. It is, however, still necessary to check the instances in which the value for the language code is `NaN`

[19]:
```
data[['title', 'overview']][data['language_code'].isna()].sample(n=10, random_
↪state=seed)
```

[19]:
```
                                                title  \
3241  Born Free: A Lioness of Two Worlds (Story of E...
3050                                       Stone Soup
4807  The Glass Magician (The Paper Magician Trilogy...
9918                     Nothing's Fair in Fifth Grade
3971  Experiencing God: Knowing and Doing the Will o...
9772  The Voyages of Doctor Dolittle (Doctor Dolittl...
8179                                       First Love
2048                      Ramona the Pest (Ramona, #2)
9559                   Relentless (The Lost Fleet, #5)
9240    Truth Will Prevail (The Work and the Glory, #3)

                                             overview
3241  There have been many accounts of the return to...
3050  First published in 1947, this classic picture ...
4807  Three months after returning Magician Emery Th...
9918  Jenny knows one thing for sure - Elsie Edwards...
3971  Most Bible studies help people; this one chang...
9772  The delightfully eccentric Doctor Dolittle, re...
8179  An extraordinary portrait of true love that wi...
2048  This is the second title in the hugely popular...
9559  After successfully freeing Alliance POWs, "Bla...
9240                                              NaN
```

More than 10% of the data has no language code. We verified that all of them are in English, thus they do not have to be deleted. Furthermore, the `language_code` feature is no longer needed.

[20]:
```
eng_lc = set(['en', 'en-CA', 'en-US', 'en-GB', 'eng'])

data = data[(data.language_code.isin(eng_lc)) | (data.language_code.isna())].drop(
↪'language_code', axis=1)
data.sample(n=10, random_state=seed)
```

[20]:
```
      gr_book_id                                  authors  \
7203      342994  Hans Christian Andersen, Rachel Isadora
8399       53200                          Stephen Hawking
8179    17899392             James Patterson, Emily Raymond
7047     2033217                             Daniel Silva
1091    17288661                             John Grisham
2050       13872                           Katherine Dunn
8558     1015311                             Ken Akamatsu
6090    21849362                               J.R. Ward
6774        7389           Brian K. Vaughan, Adrian Alphona
5760      522525            Carol Tavris, Elliot Aronson

                                                title  \
7203                             The Little Match Girl
8399                    Black Holes and Baby Universes
8179                                       First Love
7047                     Moscow Rules (Gabriel Allon, #8)
1091                                     Sycamore Row
2050                                        Geek Love
8558                              Love Hina, Vol. 01
6090         The Shadows (Black Dagger Brotherhood, #13)
6774     Runaways, Vol. 1: Pride and Joy (Runaways, #1)
```

(continues on next page)

(continued from previous page)

```
5760  Mistakes Were Made (But Not by Me): Why We Jus...

                                              overview
7203  The wares of the poor little match girl illumi...
8399  NY Times bestseller. 13 extraordinary essays s...
8179  An extraordinary portrait of true love that wi...
7047  Now the death of a journalist leads Allon to R...
1091  Seth Hubbard is a wealthy man dying of lung ca...
2050  Geek Love is the story of the Binewskis, a car...
8558  At the age of 5, Keitaro and his childhood swe...
6090  Trez "Latimer" doesn't really exist. And not j...
6774  Meet Alex, Karolina, Gert, Chase, Molly and Ni...
5760  Why do people dodge responsibility when things...
```

6.6 Remove noisy data from book titles.

The nomenclature utilized for the book titles is as follows: $book\_title + (book\_saga\_name\ \#N\_book\_saga)$. Both the book title and the book saga can be valuable information. However, the book saga number, along with the # symbol may be removed. I have defined a method called `clean_book_title` that allows removing either all saga information or just the saga number.

```
[21]: data.title = [utils.clean_book_title(title) for title in data.title.tolist()]
      data.title.sample(n=10, random_state=seed)
```

```
[21]: 7203                                  The Little Match Girl
      8399                         Black Holes and Baby Universes
      8179                                             First Love
      7047                           Moscow Rules (Gabriel Allon)
      1091                                           Sycamore Row
      2050                                              Geek Love
      8558                                    Love Hina, Vol. 01
      6090                  The Shadows (Black Dagger Brotherhood)
      6774              Runaways, Vol. 1: Pride and Joy (Runaways)
      5760    Mistakes Were Made (But Not by Me): Why We Jus...
      Name: title, dtype: object
```

6.7 Remove noisy data from book overviews.

Luckily, text is automatically tokenized before being fed into any transformer model. That notwithstanding, there is still some work we need to do to clean our text beforehand, like removing special characters, removing extra blank spaces, etc. The `maketrans` built-in method comes handy. It enables us to create a mapping table. We can create an empty mapping table, but the third argument of this function allows us to list all of the characters to remove during the translation process. On the other hand, we will use the `re` module to work with regular expressions with python to further fix some wrong text patterns.

For further details, please check the implementation included in the `utils` module for the `clean_overview` method. Let's see an example:

```
[22]: text = data.overview[data.gr_book_id == 5354].tolist()[0]
      print(f'BEFORE cleaning:\n {text}\n\nAFTER cleaning:\n{utils.clean_overview(text)}')
```

```
BEFORE cleaning:
 Trumble is a minimum-security federal prison, a "camp," home to the usual assortment␣
↪of relatively harmless criminals--drug dealers, bank robbers, swindlers, embezzlers,
↪ tax evaders, two Wall Street crooks, one doctor, at least five lawyers.And three␣
↪former judges who call themselves the Brethren: one from Texas, one from California,
↪ and one from Mississippi. They meet each day in the law library, their turf at␣
↪Trumble, where they write briefs, handle cases for other inmates, practice law␣
↪without a license, and sometimes dispense jailhouse justice. And they spend hours␣
↪writing letters. They are fine-tuning a mail scam, and it's starting to really work.
↪ The money is pouring in.Then their little scam goes awry. It ensnares the wrong␣
↪victim, a powerful man on the outside, a man with dangerous friends, and the␣
↪Brethren's days of quietly marking time are over.

AFTER cleaning:
Trumble is a minimum-security federal prison, a camp, home to the usual assortment of␣
↪relatively harmless criminals drug dealers, bank robbers, swindlers, embezzlers,␣
↪tax evaders, two Wall Street crooks, one doctor, at least five lawyers. And three␣
↪former judges who call themselves the Brethren: one from Texas, one from California,
↪ and one from Mississippi. They meet each day in the law library, their turf at␣
↪Trumble, where they write briefs, handle cases for other inmates, practice law␣
↪without a license, and sometimes dispense jailhouse justice. And they spend hours␣
↪writing letters. They are fine-tuning a mail scam, and it's starting to really work.
↪ The money is pouring in. Then their little scam goes awry. It ensnares the wrong␣
↪victim, a powerful man on the outside, a man with dangerous friends, and the␣
↪Brethren's days of quietly marking time are over.
```

[23]:
```
data.overview = [utils.clean_overview(str(overview)) for overview in data.overview.
↪tolist()]
```

Once the preprocessing is done, the dataframe can be exported to a CSV file to avoid repeating these steps everytime we need to work with the cleaned data.

[24]:
```
data.set_index('gr_book_id').to_csv(DIR_DATASET + 'books_processed.csv', sep=',')
```

6.8 Annex. Code to perform data scraping.

The webpage for each book follow the format `https://www.goodreads.com/book/show/book_id`. For instance, https://www.goodreads.com/book/show/320 is the page containing information for the book "One Hundred Years of Solitude" by Gabriel García Márquez.

The overview is contained in an object called `readable stacked` that can be seen inspecting the code of the page.

[ ]:
```
import requests
from bs4 import BeautifulSoup

def scrap_book_overview(book_id, save=False):
    try:
        # Connect to the page
        url = "https://www.goodreads.com/book/show/"+str(book_id)
        response = requests.get(url)
        # Instantiate a BeautifulSoup object.
        soup = BeautifulSoup(response.text, 'lxml')
        # Access to the component
        sec = soup.find("div", {"class": "readable stacked"})
```

(continues on next page)

(continued from previous page)

```
        # Extract the overview
        overview = sec.findAll('span')[-1]
        # Store it, should you require it
        if not overview.text is None and save:
            file = open("overviews/"+str(book_id)+".txt","w")
            file.write(overview.text)
            file.close()
        return overview.text
    except:
        return None
```

CHAPTER SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

L

M