Github repo for the original paper: https://github.com/MaartenGr/BERTopic_evaluation/tree/main (https://github.com/MaartenGr/BERTopic_evaluation/tree/main)

- All evaluations are carried out in Kaggle notebook thus were contrained by the Kaggle's 12-hour training limit, thus we could not finish running LDA sequence model. Similarly, CTM evaluations for 20NewsGroup and BBC News datasets are also skipped.
- We skipped all models using `Top2Vec` or `Doc2Vec` models due to unsolved conflicts caused by the older version of the `gensim` library.
- We did not test Wall Time of models as packages/libraries have changed significantly in the last two years, so the results won't be comparable with the original paper.

In [2]:
```
!pip install "cython<3.0.0"
```

```
Requirement already satisfied: cython<3.0.0 in /opt/conda/lib/python3.10/site-package
s (0.29.37)
```

In [3]:
```
!pip install --no-build-isolation pyyaml==5.4.1  # Try installing pyyaml again
```

```
Collecting pyyaml==5.4.1
  Downloading PyYAML-5.4.1.tar.gz (175 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1
75.1/175.1 kB 4.3 MB/s eta 0:00:0000:01
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: pyyaml
  Building wheel for pyyaml (pyproject.toml) ... done
  Created wheel for pyyaml: filename=PyYAML-5.4.1-cp310-cp310-linux_x86_64.whl size=1
55376 sha256=788c7f8c27cdec6c7cd5b13f253ce9520cdc0b976385c2cd864236be1607131d
  Stored in directory: /root/.cache/pip/wheels/c7/0d/22/696ee92245ad710f506eee79bb05c
740d8abccd3ecdb778683
Successfully built pyyaml
Installing collected packages: pyyaml
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 6.0.1
    Uninstalling PyYAML-6.0.1:
      Successfully uninstalled PyYAML-6.0.1
ERROR: pip's dependency resolver does not currently take into account all the package
s that are installed. This behaviour is the source of the following dependency confli
cts.
jupyterlab 4.2.3 requires jupyter-lsp>=2.0.0, but you have jupyter-lsp 1.5.1 which is
incompatible.
jupyterlab-lsp 5.1.0 requires jupyter-lsp>=2.0.0, but you have jupyter-lsp 1.5.1 whic
h is incompatible.
kfp 2.5.0 requires google-cloud-storage<3,>=2.2.1, but you have google-cloud-storage
1.44.0 which is incompatible.
ydata-profiling 4.6.4 requires numpy<1.26,>=1.16.0, but you have numpy 1.26.4 which i
s incompatible.
Successfully installed pyyaml-5.4.1
```

In [4]:

```
# !pip install wurlitzer
# !pip install keras==2.15.0
!pip install pytest>=5.4.3, pytest-cov>=2.6.1
!pip install mkdocs>=1.1, mkdocs-material>=4.6.3, mkdocstrings>=0.8.0
!pip install nltk>=3.2.4, srsly>=1.0.5
#!pip install octis #==1.10.2 # - original paper
!pip install contextualized_topic_models==2.2.1
```

```
ERROR: pip's dependency resolver does not currently take into account all the pack
ages that are installed. This behaviour is the source of the following dependency
conflicts.
tensorflow-decision-forests 1.8.1 requires wurlitzer, which is not installed.
tensorflow 2.15.0 requires keras<2.16,>=2.15.0, but you have keras 3.4.1 which is
incompatible.
Collecting contextualized_topic_models==2.2.1
  Downloading contextualized_topic_models-2.2.1-py2.py3-none-any.whl.metadata (23
kB)
Requirement already satisfied: numpy>=1.19.1 in /opt/conda/lib/python3.10/site-pac
kages (from contextualized_topic_models==2.2.1) (1.26.4)
Requirement already satisfied: torchvision>=0.7.0 in /opt/conda/lib/python3.10/sit
e-packages (from contextualized_topic_models==2.2.1) (0.16.2+cpu)
Requirement already satisfied: torch>=1.6.0 in /opt/conda/lib/python3.10/site-pack
ages (from contextualized_topic_models==2.2.1) (2.1.2+cpu)
Requirement already satisfied: gensim>=3.8.3 in /opt/conda/lib/python3.10/site-pac
kages (from contextualized_topic_models==2.2.1) (4.3.2)
Collecting sentence-transformers>=1.1.1 (from contextualized_topic_models==2.2.1)
  Downloading sentence_transformers-3.0.1-py3-none-any.whl.metadata (10 kB)
[illegible]
```

```
In [5]: !pip install bertopic==0.9.4  # 0.9.4
```

```
Collecting bertopic==0.9.4
  Downloading bertopic-0.9.4-py2.py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: numpy>=1.20.0 in /opt/conda/lib/python3.10/site-packag
es (from bertopic==0.9.4) (1.26.4)
Collecting hdbscan>=0.8.27 (from bertopic==0.9.4)
  Downloading hdbscan-0.8.37-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl.metadata (13 kB)
Requirement already satisfied: umap-learn>=0.5.0 in /opt/conda/lib/python3.10/site-pa
ckages (from bertopic==0.9.4) (0.5.6)
Requirement already satisfied: pandas>=1.1.5 in /opt/conda/lib/python3.10/site-packag
es (from bertopic==0.9.4) (2.2.2)
Requirement already satisfied: scikit-learn>=0.22.2.post1 in /opt/conda/lib/python3.1
0/site-packages (from bertopic==0.9.4) (1.2.2)
Requirement already satisfied: tqdm>=4.41.1 in /opt/conda/lib/python3.10/site-package
s (from bertopic==0.9.4) (4.66.4)
Requirement already satisfied: sentence-transformers>=0.4.1 in /opt/conda/lib/python
3.10/site-packages (from bertopic==0.9.4) (3.0.1)
Requirement already satisfied: plotly>=4.7.0 in /opt/conda/lib/python3.10/site-packag
es (from bertopic==0.9.4) (5.18.0)
Requirement already satisfied: pyyaml<6.0 in /opt/conda/lib/python3.10/site-packages
(from bertopic==0.9.4) (5.4.1)
Requirement already satisfied: cython<3,>=0.27 in /opt/conda/lib/python3.10/site-pack
ages (from hdbscan>=0.8.27->bertopic==0.9.4) (0.29.37)
Requirement already satisfied: scipy>=1.0 in /opt/conda/lib/python3.10/site-packages
(from hdbscan>=0.8.27->bertopic==0.9.4) (1.11.4)
Requirement already satisfied: joblib>=1.0 in /opt/conda/lib/python3.10/site-packages
(from hdbscan>=0.8.27->bertopic==0.9.4) (1.4.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.10/si
te-packages (from pandas>=1.1.5->bertopic==0.9.4) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-package
s (from pandas>=1.1.5->bertopic==0.9.4) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.10/site-packa
ges (from pandas>=1.1.5->bertopic==0.9.4) (2023.4)
Requirement already satisfied: tenacity>=6.2.0 in /opt/conda/lib/python3.10/site-pack
ages (from plotly>=4.7.0->bertopic==0.9.4) (8.2.3)
Requirement already satisfied: packaging in /opt/conda/lib/python3.10/site-packages
(from plotly>=4.7.0->bertopic==0.9.4) (21.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.10/site
-packages (from scikit-learn>=0.22.2.post1->bertopic==0.9.4) (3.2.0)
Requirement already satisfied: transformers<5.0.0,>=4.34.0 in /opt/conda/lib/python3.
10/site-packages (from sentence-transformers>=0.4.1->bertopic==0.9.4) (4.42.3)
Requirement already satisfied: torch>=1.11.0 in /opt/conda/lib/python3.10/site-packag
es (from sentence-transformers>=0.4.1->bertopic==0.9.4) (2.1.2+cpu)
Requirement already satisfied: huggingface-hub>=0.15.1 in /opt/conda/lib/python3.10/s
ite-packages (from sentence-transformers>=0.4.1->bertopic==0.9.4) (0.23.4)
Requirement already satisfied: Pillow in /opt/conda/lib/python3.10/site-packages (fro
m sentence-transformers>=0.4.1->bertopic==0.9.4) (9.5.0)
Requirement already satisfied: numba>=0.51.2 in /opt/conda/lib/python3.10/site-packag
es (from umap-learn>=0.5.0->bertopic==0.9.4) (0.58.1)
Requirement already satisfied: pynndescent>=0.5 in /opt/conda/lib/python3.10/site-pac
kages (from umap-learn>=0.5.0->bertopic==0.9.4) (0.5.13)
Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-packages (f
rom huggingface-hub>=0.15.1->sentence-transformers>=0.4.1->bertopic==0.9.4) (3.13.1)
Requirement already satisfied: fsspec>=2023.5.0 in /opt/conda/lib/python3.10/site-pac
kages (from huggingface-hub>=0.15.1->sentence-transformers>=0.4.1->bertopic==0.9.4)
(2024.5.0)
Requirement already satisfied: requests in /opt/conda/lib/python3.10/site-packages (f
rom huggingface-hub>=0.15.1->sentence-transformers>=0.4.1->bertopic==0.9.4) (2.32.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/conda/lib/python3.1
0/site-packages (from huggingface-hub>=0.15.1->sentence-transformers>=0.4.1->bertopic
==0.9.4) (4.9.0)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /opt/conda/lib/python3.1
0/site-packages (from numba>=0.51.2->umap-learn>=0.5.0->bertopic==0.9.4) (0.41.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/lib/python3.10/
site-packages (from packaging->plotly>=4.7.0->bertopic==0.9.4) (3.1.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-packages (f
rom python-dateutil>=2.8.2->pandas>=1.1.5->bertopic==0.9.4) (1.16.0)
Requirement already satisfied: sympy in /opt/conda/lib/python3.10/site-packages (from
torch>=1.11.0->sentence-transformers>=0.4.1->bertopic==0.9.4) (1.13.0)
Requirement already satisfied: networkx in /opt/conda/lib/python3.10/site-packages (f
rom torch>=1.11.0->sentence-transformers>=0.4.1->bertopic==0.9.4) (3.2.1)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.10/site-packages (fro
m torch>=1.11.0->sentence-transformers>=0.4.1->bertopic==0.9.4) (3.1.2)
Requirement already satisfied: regex!=2019.12.17 in /opt/conda/lib/python3.10/site-pa
ckages (from transformers<5.0.0,>=4.34.0->sentence-transformers>=0.4.1->bertopic==0.
9.4) (2023.12.25)
Requirement already satisfied: safetensors>=0.4.1 in /opt/conda/lib/python3.10/site-p
ackages (from transformers<5.0.0,>=4.34.0->sentence-transformers>=0.4.1->bertopic==0.
9.4) (0.4.3)
Requirement already satisfied: tokenizers<0.20,>=0.19 in /opt/conda/lib/python3.10/si
te-packages (from transformers<5.0.0,>=4.34.0->sentence-transformers>=0.4.1->bertopic
==0.9.4) (0.19.1)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.10/site-pack
ages (from jinja2->torch>=1.11.0->sentence-transformers>=0.4.1->bertopic==0.9.4) (2.
1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.10/
site-packages (from requests->huggingface-hub>=0.15.1->sentence-transformers>=0.4.1->
bertopic==0.9.4) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-package
s (from requests->huggingface-hub>=0.15.1->sentence-transformers>=0.4.1->bertopic==0.
9.4) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.10/site-p
ackages (from requests->huggingface-hub>=0.15.1->sentence-transformers>=0.4.1->bertop
ic==0.9.4) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/site-p
ackages (from requests->huggingface-hub>=0.15.1->sentence-transformers>=0.4.1->bertop
ic==0.9.4) (2024.7.4)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /opt/conda/lib/python3.10/site-p
ackages (from sympy->torch>=1.11.0->sentence-transformers>=0.4.1->bertopic==0.9.4)
(1.3.0)
Downloading bertopic-0.9.4-py2.py3-none-any.whl (57 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5
7.6/57.6 kB 2.6 MB/s eta 0:00:00
Downloading hdbscan-0.8.37-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(3.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
3.6/3.6 MB 47.1 MB/s eta 0:00:0000:0100:01
Installing collected packages: hdbscan, bertopic
Successfully installed bertopic-0.9.4 hdbscan-0.8.37
```

In [5]:
```
# !pip install top2vec==1.0.26 #==1.0.26 1.0.34
```

In [6]:
```
!pip install octis
```

```
Collecting octis
  Downloading octis-1.14.0-py2.py3-none-any.whl.metadata (27 kB)
Requirement already satisfied: gensim<5.0,>=4.2.0 in /opt/conda/lib/python3.10/sit
e-packages (from octis) (4.3.2)
Requirement already satisfied: nltk in /opt/conda/lib/python3.10/site-packages (fr
om octis) (3.2.4)
Requirement already satisfied: pandas in /opt/conda/lib/python3.10/site-packages
(from octis) (2.2.2)
Requirement already satisfied: spacy in /opt/conda/lib/python3.10/site-packages (f
rom octis) (3.7.5)
Collecting scikit-learn==1.1.0 (from octis)
  Downloading scikit_learn-1.1.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x
86_64.whl.metadata (10 kB)
Requirement already satisfied: scikit-optimize>=0.8.1 in /opt/conda/lib/python3.1
0/site-packages (from octis) (0.10.2)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.10/site-packag
es (from octis) (3.7.5)
Requirement already satisfied: torch in /opt/conda/lib/python3.10/site-packages (f
rom octis) (2.1.2+cpu)
```

In [7]:
```
import gc
gc.collect()
```

Out[7]:
```
11
```

In [8]:
~~~python
import re
import nltk
import string
import pandas as pd
import numpy as np

from typing import List, Tuple, Union
from octis.dataset.dataset import Dataset
from octis.preprocessing.preprocessing import Preprocessing

nltk.download("punkt")


class DataLoader:
    """Prepare and load custom data using OCTIS

    Arguments:
        dataset: The name of the dataset, default options:
                    * trump
                    * 20news

    Usage:

    **Trump** - Unprocessed

    ```python
    from evaluation import DataLoader
    dataloader = DataLoader(dataset="trump").prepare_docs(save="trump.txt").preprocess_c
    ~~~

    20 Newsgroups - Unprocessed

    ```python
    from evaluation import DataLoader
    dataloader = DataLoader(dataset="20news").prepare_docs(save="20news.txt").preprocess
    ```

    Custom Data

    Whenever you want to use a custom dataset (list of strings), make sure to use the lo

    ```python
    from evaluation import DataLoader
    dataloader = DataLoader(dataset="my_docs").prepare_docs(save="my_docs.txt", docs=my_
    ```
    """

    def __init__(self, dataset: str):
        self.dataset = dataset
        self.docs = None
        self.timestamps = None
        self.octis_docs = None
        self.doc_path = None

    def load_docs(
        self, save: bool = False, docs: List[str] = None
) -> Tuple[List[str], Union[List[str], None]]:
```
```

~~~python
        """Load in the documents

        ```python
        dataloader = DataLoader(dataset="trump")
        docs, timestamps = dataloader.load_docs()
        ~~~
        """
        if docs is not None:
            return self.docs, None

        if self.dataset == "trump":
            self.docs, self.timestamps = self._trump()
        elif self.dataset == "trump_dtm":
            self.docs, self.timestamps = self._trump_dtm()
        elif self.dataset == "un_dtm":
            self.docs, self.timestamps = self._un_dtm()
        elif self.dataset == "20news":
            self.docs, self.timestamps = self._20news()

        if save:
            self._save(self.docs, save)

        return self.docs, self.timestamps

    def load_octis(self, custom: bool = False) -> Dataset:
        """Get dataset from OCTIS

        Arguments:
            custom: Whether a custom dataset is used or one retrieved from
                    https://github.com/MIND-Lab/OCTIS#available-datasets

        Usage:

        ```python
        from evaluation import DataLoader
        dataloader = DataLoader(dataset="20news")
        data = dataloader.load_octis(custom=True)
        ```
        """
        data = Dataset()

        if custom:
            data.load_custom_dataset_from_folder(self.dataset)
        else:
            data.fetch_dataset(self.dataset)

        self.octis_docs = data
        return self.octis_docs

    def prepare_docs(self, save: bool = False, docs: List[str] = None):
        """Prepare documents

        Arguments:
            save: The path to save the model to, make sure it ends in .json
            docs: The documents you want to preprocess in OCTIS

        Usage:
```
```

~~~
        ```python
        from evaluation import DataLoader
        dataloader = DataLoader(dataset="my_docs").prepare_docs(save="my_docs.txt", docs
        ~~~
        """
        self.load_docs(save, docs)
        return self

    def preprocess_octis(
        self,
        preprocessor: Preprocessing = None,
        documents_path: str = None,
        output_folder: str = "docs",
    ):
        """Preprocess the data using OCTIS

        Arguments:
            preprocessor: Custom OCTIS preprocessor
            documents_path: Path to the .txt file
            output_folder: Path to where you want to save the preprocessed data

        Usage:

        ```python
        from evaluation import DataLoader
        dataloader = DataLoader(dataset="my_docs").prepare_docs(save="my_docs.txt", docs
        dataloader.preprocess_octis(output_folder="my_docs")
        ```

        If you want to use your custom preprocessor:

        ```python
        from evaluation import DataLoader
        from octis.preprocessing.preprocessing import Preprocessing

        preprocessor = Preprocessing(lowercase=False,
                                     remove_punctuation=False,
                                     punctuation=string.punctuation,
                                     remove_numbers=False,
                                     lemmatize=False,
                                     language='english',
                                     split=False,
                                     verbose=True,
                                     save_original_indexes=True,
                                     remove_stopwords_spacy=False)

        dataloader = DataLoader(dataset="my_docs").prepare_docs(save="my_docs.txt", docs
        dataloader.preprocess_octis(preprocessor=preprocessor, output_folder="my_docs")
        ```
        """
        if preprocessor is None:
            preprocessor = Preprocessing(
                lowercase=False,
                remove_punctuation=False,
                punctuation=string.punctuation,
                remove_numbers=False,
```
```

```
                lemmatize=False,
                language="english",
                split=False,
                verbose=True,
                save_original_indexes=True,
                remove_stopwords_spacy=False,
            )
        if not documents_path:
            documents_path = self.doc_path
        dataset = preprocessor.preprocess_dataset(documents_path=documents_path)
        dataset.save(output_folder)

    def _trump(self) -> Tuple[List[str], List[str]]:
        """Prepare the trump dataset"""
        trump = pd.read_csv(
            "https://drive.google.com/uc?export=download&id=1xRKHaP-QwACMydlDnyFPEaFdtsk
        )
        trump = trump.loc[(trump.isRetweet == "f") & (trump.text != ""), :]
        timestamps = trump.date.to_list()
        docs = trump.text.to_list()
        docs = [doc.lower().replace("\n", " ") for doc in docs if len(doc) > 2]
        timestamps = [
            timestamp for timestamp, doc in zip(timestamps, docs) if len(doc) > 2
        ]
        return docs, timestamps

    def _trump_dtm(self) -> Tuple[List[str], List[str]]:
        """Prepare the trump dataset including timestamps"""
        trump = pd.read_csv(
            "https://drive.google.com/uc?export=download&id=1xRKHaP-QwACMydlDnyFPEaFdtsk
        )
        trump = trump.loc[(trump.isRetweet == "f") & (trump.text != ""), :]
        timestamps = trump.date.to_list()
        documents = trump.text.to_list()

        docs = []
        time = []
        for doc, timestamp in zip(documents, timestamps):
            if len(doc) > 2:
                docs.append(doc.lower().replace("\n", " "))
                time.append(timestamp)

        # Create bins
        nr_bins = 10
        df = pd.DataFrame({"Doc": docs, "Timestamp": time}).sort_values("Timestamp")
        df["Timestamp"] = pd.to_datetime(df["Timestamp"], infer_datetime_format=True)
        df["Bins"] = pd.cut(df.Timestamp, bins=nr_bins)
        df["Timestamp"] = df.apply(lambda row: row.Bins.left, 1)
        timestamps = df.Timestamp.tolist()
        documents = df.Doc.tolist()

        return docs, timestamps

    def _un_dtm(self) -> Tuple[List[str], List[str]]:
        """Prepare the UN dataset"""

        def create_paragraphs(text):
            text = text.replace("Mr.\n", "Mr. ")
            text = text.replace(".\n", " \p ")
            text = text.replace(". \n ", " \p ")
            text = text.replace(". \n", " \p ")
            text = text.replace("\n", " ")
            text = [x.strip().lower() for x in text.split("\p")]
            return text

        dataset = pd.read_csv(
            "https://runestone.academy/runestone/books/published/httlads/_static/un-gene
        )
        dataset["text"] = dataset.apply(lambda row: create_paragraphs(row.text), 1)
        dataset = dataset.explode("text").sort_values("year")
        dataset = dataset.loc[dataset.year > 2005, :] # original: > 2005
        # Set a random seed for reproducibility
        np.random.seed(42)

        # Define a function to sample at most 2000 entries per year
        def sample_yearly(df, year_column, max_samples):
            return df.groupby(year_column).apply(lambda x: x.sample(min(len(x), max_sa

        dataset = sample_yearly(dataset, 'year', 2000)

        docs = dataset.text.tolist()
        timestamps = dataset.year.tolist()
        return docs, timestamps

    def _save(self, docs: List[str], save: str):
        """Save the documents"""
        with open(save, mode="wt", encoding="utf-8") as myfile:
            myfile.write("\n".join(docs))

        self.doc_path = save
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Evaluations

In [9]:
```
import json
import time
import itertools
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer
from typing import Mapping, Any, List, Tuple

try:
    from bertopic import BERTopic
except ImportError:
    pass

try:
    from top2vec import Top2Vec
except ImportError:
    pass

try:
    from contextualized_topic_models.models.ctm import CombinedTM
    from contextualized_topic_models.utils.data_preparation import (
        TopicModelDataPreparation,
    )
    import nltk

    nltk.download("stopwords")
    from nltk.corpus import stopwords
except ImportError:
    pass

from octis.models.ETM import ETM
from octis.models.LDA import LDA
from octis.models.NMF import NMF
from octis.models.CTM import CTM
from octis.dataset.dataset import Dataset
from octis.evaluation_metrics.diversity_metrics import TopicDiversity
from octis.evaluation_metrics.coherence_metrics import Coherence

import gensim
import gensim.corpora as corpora
from gensim.models import ldaseqmodel


class Trainer:

    def __init__(
        self,
        dataset: str,
        model_name: str,
        params: Mapping[str, Any],
        topk: int = 10,
        custom_dataset: bool = False,
        bt_embeddings: np.ndarray = None,
        bt_timestamps: List[str] = None,
        bt_nr_bins: int = None,
        custom_model=None,
        verbose: bool = True,
```

~~~python
):
    self.dataset = dataset
    self.custom_dataset = custom_dataset
    self.model_name = model_name
    self.params = params
    self.topk = topk
    self.timestamps = bt_timestamps
    self.nr_bins = bt_nr_bins
    self.embeddings = bt_embeddings
    self.ctm_preprocessed_docs = None
    self.custom_model = custom_model
    self.verbose = verbose

    # Prepare data and metrics
    self.data = self.get_dataset()
    self.metrics = self.get_metrics()

    # CTM
    self.qt_ctm = None
    self.training_dataset_ctm = None

def train(self, save: str = False) -> Mapping[str, Any]:
    """Train a topic model

    Arguments:
        save: The name of the file to save it to.
              It will be saved as a .json in the current
              working directory

    Usage:

    ```python
    from evaluation import Trainer
    dataset, custom = "20NewsGroup", False
    params = {"num_topics": [(i+1)*10 for i in range(5)], "random_state": 42}

    trainer = Trainer(dataset=dataset,
                      model_name="LDA",
                      params=params,
                      custom_dataset=custom,
                      verbose=True)
    results = trainer.train(save="LDA_results")
    ~~~
    """

    results = []

    # Loop over all parameters
    params_name = list(self.params.keys())
    params = {
        param: (value if type(value) == list else [value])
        for param, value in self.params.items()
    }
    new_params = list(itertools.product(*params.values()))
    for param_combo in new_params:

        # Train and evaluate model
```
```

```
            params_to_use = {
                param: value for param, value in zip(params_name, param_combo)
            }
            output, computation_time = self._train_tm_model(params_to_use)
            scores = self.evaluate(output)

            # Update results
            result = {
                "Dataset": self.dataset,
                "Dataset Size": len(self.data.get_corpus()),
                "Model": self.model_name,
                "Params": params_to_use,
                "Scores": scores,
                "Computation Time": computation_time,
            }
            results.append(result)

        if save:
            with open(f"{save}.json", "w") as f:
                json.dump(results, f)

            try:
                from google.colab import files

                files.download(f"{save}.json")
            except ImportError:
                pass

        return results

    def _train_tm_model(
        self, params: Mapping[str, Any]
    ) -> Tuple[Mapping[str, Any], float]:
        """Select and train the Topic Model"""
        # Train custom CTM
        if self.model_name == "CTM_CUSTOM":
            if self.qt_ctm is None:
                self._preprocess_ctm()
            return self._train_ctm(params)

        # Train BERTopic
        elif "BERTopic" in self.model_name: # MODIFIED
            return self._train_bertopic(params)

        # Train Top2Vec
        elif self.model_name == "Top2Vec":
            return self._train_top2vec(params)

        # Train LDAseq
        elif self.model_name == "LDAseq":
            return self._train_ldaseq(params)

        # Train OCTIS model
        octis_models = ["ETM", "LDA", "CTM", "NMF"]
        if self.model_name in octis_models:
            return self._train_octis_model(params)
```

```
def _train_ldaseq(
    self, params: Mapping[str, any]
) -> Tuple[Mapping[str, Any], float]:
    """Train LDA seq model"""
    data = self.data.get_corpus()
    docs = [" ".join(words) for words in data]

    df = pd.DataFrame({"Doc": docs, "Timestamp": self.timestamps}).sort_values(
        "Timestamp"
    )
    df["Bins"] = pd.cut(df.Timestamp, bins=params["nr_bins"])
    df["Timestamp"] = df.apply(lambda row: row.Bins.left, 1)
    timestamps = df.groupby("Bins").count().Timestamp.values
    docs = df.Doc.values

    data_words = list(sent_to_words(docs))
    id2word = corpora.Dictionary(data_words)
    corpus = [id2word.doc2bow(text) for text in data_words]

    print(len(corpus), len(self.timestamps), timestamps)

    params["corpus"] = corpus
    params["id2word"] = id2word
    params["time_slice"] = timestamps
    del params["nr_bins"]

    import logging
    from gensim.corpora.dictionary import Dictionary
    logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=l

    start = time.time()
    ldaseq = ldaseqmodel.LdaSeqModel(**params)

    # Manually track and log time slices
    current_time_slice = 0
    doc_counter = 0
    for doc in corpus:
        doc_counter += 1
        if doc_counter > sum(timestamps[:current_time_slice+1]):
            current_time_slice += 1

        logging.info(f'Processing document {doc_counter}, current time slice: {curre


    end = time.time()
    computation_time = end - start

    all_topics = {}
    for i in range(len(timestamps)):
        topics = ldaseq.print_topics(time=i)
        topics = [[word for word, _ in topic][:5] for topic in topics]
        all_topics[i] = {"topics": topics}

    return all_topics, computation_time

def _train_top2vec(
    self, params: Mapping[str, Any]
) -> Tuple[Mapping[str, Any], float]:
    """Train Top2Vec"""
    nr_topics = None
    data = self.data.get_corpus()
    data = [" ".join(words) for words in data]
    params["documents"] = data

    if params.get("nr_topics"):
        nr_topics = params["nr_topics"]
        del params["nr_topics"]

    start = time.time()

    if self.custom_model is not None:
        model = self.custom_model(**params)
    else:
        model = Top2Vec(**params)

    if nr_topics:
        try:
            _ = model.hierarchical_topic_reduction(nr_topics)
            params["reduction"] = True
            params["nr_topics"] = nr_topics
        except:
            params["reduction"] = False
            nr_topics = False

    end = time.time()
    computation_time = float(end - start)

    if nr_topics:
        topic_words, _, _ = model.get_topics(reduced=True)
    else:
        topic_words, _, _ = model.get_topics(reduced=False)

    topics_old = [list(topic[:10]) for topic in topic_words]
    all_words = [word for words in self.data.get_corpus() for word in words]
    topics = []
    for topic in topics_old:
        words = []
        for word in topic:
            if word in all_words:
                words.append(word)
            else:
                print(f"error: {word}")
                words.append(all_words[0])
        topics.append(words)

    if not nr_topics:
        params["nr_topics"] = len(topics)
        params["reduction"] = False

    del params["documents"]
    output_tm = {
        "topics": topics,
    }
    return output_tm, computation_time
def _train_ctm(self, params) -> Tuple[Mapping[str, Any], float]:
    """Train CTM"""
    params["bow_size"] = len(self.qt_ctm.vocab)
    ctm = CombinedTM(**params)

    start = time.time()
    ctm.fit(self.training_dataset_ctm)
    end = time.time()
    computation_time = float(end - start)

    topics = ctm.get_topics(10)
    topics = [topics[x] for x in topics]

    output_tm = {
        "topics": topics,
    }

    return output_tm, computation_time

def _preprocess_ctm(self):
    """Preprocess data for CTM"""
    # Prepare docs
    data = self.data.get_corpus()
    docs = [" ".join(words) for words in data]

    # Remove stop words
    stop_words = stopwords.words("english")
    preprocessed_documents = [
        " ".join([x for x in doc.split(" ") if x not in stop_words]).strip()
        for doc in docs
    ]

    # Get vocabulary
    vectorizer = CountVectorizer(
        max_features=2000, token_pattern=r"\b[a-zA-Z]{2,}\b"
    )
    vectorizer.fit_transform(preprocessed_documents)
    # vocabulary = set(vectorizer.get_feature_names())
    try:
        vocabulary = set(vectorizer.get_feature_names_out())
    except AttributeError:
        vocabulary = set(vectorizer.get_feature_names())

    # Preprocess documents further
    preprocessed_documents = [
        " ".join([w for w in doc.split() if w in vocabulary]).strip()
        for doc in preprocessed_documents
    ]

    # Prepare CTM data
    qt = TopicModelDataPreparation("all-mpnet-base-v2")
    training_dataset = qt.fit(
        text_for_contextual=docs, text_for_bow=preprocessed_documents
    )

    self.qt_ctm = qt
    self.training_dataset_ctm = training_dataset

def _train_octis_model(
    self, params: Mapping[str, any]
) -> Tuple[Mapping[str, Any], float]:
    """Train OCTIS model"""

    if self.model_name == "ETM":
        model = ETM(**params)
        model.use_partitions = False
    elif self.model_name == "LDA":
        model = LDA(**params)
        model.use_partitions = False
    elif self.model_name == "CTM":
        model = CTM(**params)
        model.use_partitions = False
    elif self.model_name == "NMF":
        model = NMF(**params)
        model.use_partitions = False

    start = time.time()
    output_tm = model.train_model(self.data)
    end = time.time()
    computation_time = end - start
    return output_tm, computation_time

def _train_bertopic(
    self, params: Mapping[str, any]
) -> Tuple[Mapping[str, Any], float]:
    """Train BERTopic model"""
    data = self.data.get_corpus()
    data = [" ".join(words) for words in data]
    params["calculate_probabilities"] = False

    if self.custom_model is not None:
        model = self.custom_model(**params)
    else:
        model = BERTopic(**params)

    start = time.time()
    topics, _ = model.fit_transform(data, self.embeddings)


    # Dynamic Topic Modeling
    if self.timestamps:
        topics_over_time = model.topics_over_time(
            data,
            topics,
            self.timestamps,
            nr_bins=self.nr_bins,
            evolution_tuning=False,
            global_tuning=False,
        )
        unique_timestamps = topics_over_time.Timestamp.unique()
        dtm_topics = {}
        for unique_timestamp in unique_timestamps:
            dtm_topic = topics_over_time.loc[
                topics_over_time.Timestamp == unique_timestamp, :
            ].sort_values("Frequency", ascending=True)
            dtm_topic = dtm_topic.loc[dtm_topic.Topic != -1, :]
            dtm_topic = [topic.split(", ") for topic in dtm_topic.Words.values]
            dtm_topics[unique_timestamp] = {"topics": dtm_topic}

            all_words = [word for words in self.data.get_corpus() for word in word

            updated_topics = []
            for topic in dtm_topic:
                updated_topic = []
                for word in topic:
                    if word not in all_words:
                        print(word)
                        updated_topic.append(all_words[0])
                    else:
                        updated_topic.append(word)
                updated_topics.append(updated_topic)

            dtm_topics[unique_timestamp] = {"topics": updated_topics}

        output_tm = dtm_topics

    end = time.time()
    computation_time = float(end - start)

    if not self.timestamps:
        all_words = [word for words in self.data.get_corpus() for word in words]
        bertopic_topics = [
            [
                vals[0] if vals[0] in all_words else all_words[0]
                for vals in model.get_topic(i)[:10]
            ]
            for i in range(len(set(topics)) - 1)
        ]

        output_tm = {"topics": bertopic_topics}

    return output_tm, computation_time

def evaluate(self, output_tm):
    """Using metrics and output of the topic model, evaluate the topic model"""
    if self.timestamps:
        results = {str(timestamp): {} for timestamp, _ in output_tm.items()}
        for timestamp, topics in output_tm.items():
            self.metrics = self.get_metrics()
            for scorers, _ in self.metrics:
                for scorer, name in scorers:
                    score = scorer.score(topics)
                    results[str(timestamp)][name] = float(score)

    else:
        # Calculate results
        results = {}
        for scorers, _ in self.metrics:
            for scorer, name in scorers:
                score = scorer.score(output_tm)
                results[name] = float(score)

        # Print results
        if self.verbose:
            print("Results")
            print("============")
            for metric, score in results.items():
                print(f"{metric}: {str(score)}")
            print(" ")

    return results

def get_dataset(self):
    """Get dataset from OCTIS"""
    data = Dataset()

    if self.custom_dataset:
        data.load_custom_dataset_from_folder(self.dataset)
    else:
        data.fetch_dataset(self.dataset)
    return data

def get_metrics(self):
    """Prepare evaluation measures using OCTIS"""
    npmi = Coherence(texts=self.data.get_corpus(), topk=self.topk, measure="c_npmi"
    topic_diversity = TopicDiversity(topk=self.topk)

    # Define methods
    coherence = [(npmi, "npmi")]
    diversity = [(topic_diversity, "diversity")]
    metrics = [(coherence, "Coherence"), (diversity, "Diversity")]

    return metrics


def sent_to_words(sentences):
for sentence in sentences:
    yield (gensim.utils.simple_preprocess(str(sentence), deacc=True))
```

```
2024-07-27 18:05:10.726655: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.c
c:9261] Unable to register cuDNN factory: Attempting to register factory for plugin c
uDNN when one has already been registered
2024-07-27 18:05:10.726835: E external/local_xla/xla/stream_executor/cuda/cuda_fft.c
c:607] Unable to register cuFFT factory: Attempting to register factory for plugin cu
FFT when one has already been registered
2024-07-27 18:05:10.889279: E external/local_xla/xla/stream_executor/cuda/cuda_blas.c
c:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered

[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Data

Trump data

In [11]:
```
%%time
dataloader = DataLoader(dataset="trump").prepare_docs(save="trump.txt").preprocess_octi
```

```
100%|██████████| 46693/46693 [00:00<00:00, 235100.83it/s]

created vocab
53637
words filtering done
CPU times: user 3.36 s, sys: 141 ms, total: 3.5 s
Wall time: 11.1 s
```

In [12]:
```
%%time
dataloader = DataLoader(dataset="trump_dtm").prepare_docs(save="trump_dtm.txt").preproc
```

```
100%|██████████| 46693/46693 [00:00<00:00, 235766.51it/s]

created vocab
53637
words filtering done
CPU times: user 5.19 s, sys: 105 ms, total: 5.29 s
Wall time: 24.1 s
```

United Nations data

In [13]:
```
%%time
dataloader = DataLoader(dataset="un_dtm").prepare_docs(save="un_dtm.txt").preprocess_oc
```

```
/tmp/ipykernel_33/3565351604.py:248: DeprecationWarning: DataFrameGroupBy.apply opera
ted on the grouping columns. This behavior is deprecated, and in a future version of
pandas the grouping columns will be excluded from the operation. Either pass `include
_groups=False` to exclude the groupings or explicitly select the grouping columns aft
er groupby to silence this warning.
  return df.groupby(year_column).apply(lambda x: x.sample(min(len(x), max_samples))).
reset_index(drop=True)
100%|██████████| 20000/20000 [00:00<00:00, 82675.95it/s]

created vocab
23029
words filtering done
CPU times: user 7.05 s, sys: 584 ms, total: 7.63 s
Wall time: 10.5 s
```

Model Evaluations

- All evaluations are carried out in Kaggle notebook thus were contrained by the Kaggle's 12-hour training limit, thus we could not finish running LDA sequence model. Similarly, CTM evaluations for 20NewsGroup and BBC News datasets are also skipped.
- We skipped all models using `Top2Vec` or `Doc2Vec` models due to unsolved conflicts caused by the older version of the `gensim` library.
- We did not test Wall Time of models as packages/libraries have changed significantly in the last two years, so the results won't be comparable with the original paper.

Trump Data

Trump - NMF(CPU)

```
In [13]: for i, random_state in enumerate([0, 21, 42]):
             print("Random State", random_state)
             dataset, custom = "trump", True
             params = {"num_topics": [(i+1)*10 for i in range(5)], "random_state": random_state

             trainer = Trainer(dataset=dataset,
                               model_name="NMF",
                               params=params,
                               custom_dataset=custom,
                               verbose=True)
             results = trainer.train(save=f"NMF_trump_{i+1}")

         print("Training COMPLETED")
```

```
Random State 0
Results
============
npmi: -0.005742839116935676
diversity: 0.4

Results
============
npmi: 0.007421120689733071
diversity: 0.42

Results
============
npmi: 0.014766098200964957
diversity: 0.3933333333333333

Results
============
npmi: 0.016208854270394913
diversity: 0.3375

Results
============
npmi: 0.017222793405267747
diversity: 0.348

Random State 21
Results
============
npmi: -0.003917059704810471
diversity: 0.46

Results
============
npmi: 0.007975363716605934
diversity: 0.415

Results
============
npmi: 0.02100240700031533
diversity: 0.38

Results
============
npmi: 0.010374395636827632
diversity: 0.36

Results
============
npmi: 0.006939519775300629
diversity: 0.316

Random State 42
Results
============
npmi: -0.004374876532201073
diversity: 0.38
```

```
Results
============
npmi: 0.006425754668690988
diversity: 0.405

Results
============
npmi: 0.011036886533741O2
diversity: 0.37666666666666665

Results
============
npmi: 0.016751179352735 63
diversity: 0.3575

Results
============
npmi: 0.015734180346554372
diversity: 0.342

Training COMPLETED
```

```
In [15]: gc.collect()
```

```
Out[15]: 32
```

Trump - LDA (CPU)

```
In [16]: for i, random_state in enumerate([0, 21, 42]):
             print("Random State", random_state)
             dataset, custom = "trump", True
             params = {"num_topics": [(i+1)*10 for i in range(5)], "random_state": random_state}

             trainer = Trainer(dataset=dataset,
                               model_name="LDA",
                               params=params,
                               custom_dataset=custom,
                               verbose=True)
             results = trainer.train(save=f"LDA_trump_{i+1}")

         print("Training COMPLETED")
```

```
Random State 0
Results
============
npmi: -0.0069114454748685815
diversity: 0.48

Results
============
npmi: -0.005003465267478509
diversity: 0.425

Results
============
npmi: -0.004333717218804874
diversity: 0.5133333333333333

Results
============
npmi: -0.013484627735113936
diversity: 0.5475

Results
============
npmi: -0.02873932987802867
diversity: 0.556

Random State 21
Results
============
npmi: -0.00873283471860894
diversity: 0.41

Results
============
npmi: -0.0032195570281142545
diversity: 0.44

Results
============
npmi: -0.011184204528511066
diversity: 0.52

Results
============
npmi: -0.010834766671676588
diversity: 0.5275

Results
============
npmi: -0.020161391882537755
diversity: 0.596

Random State 42
Results
============
npmi: -0.004250980433560532
diversity: 0.45
```

```
Results
============
npmi: -0.006889694250977099
diversity: 0.47

Results
============
npmi: -0.005643503573800876
diversity: 0.48333333333333334

Results
============
npmi: -0.011009338092369111
diversity: 0.55

Results
============
npmi: -0.018123237177004415
diversity: 0.574

Training COMPLETED
```

Trump - CTM (GPU)

For CTM, we only trained it twice due to the 12-hour training limit on Kaggle.

In [17]:
```
if not hasattr(CountVectorizer, 'get_feature_names'):
    CountVectorizer.get_feature_names = CountVectorizer.get_feature_names_out
```

In [ ]:
```
for i in range(2):
    print("*"*60)
    print("Round", i)
    dataset, custom = "trump", True
    params = {
        "n_components": [(i+1)*10 for i in range(5)],
        "contextual_size":768
    }

    trainer = Trainer(dataset=dataset,
                      model_name="CTM_CUSTOM",
                      params=params,
                      custom_dataset=custom,
                      verbose=True)
    results = trainer.train(save=f"CTM_trump_{i+1}")

print("Training COMPLETED")
```

Trump - BERTopic (CPU): `all-mpnet-base-v2`

In [17]:
```
%%capture
from sentence_transformers import SentenceTransformer

# Prepare data
dataset, custom = "trump", True
data_loader = DataLoader(dataset)
_, timestamps = data_loader.load_docs()
data = data_loader.load_octis(custom)
data = [" ".join(words) for words in data.get_corpus()]

# Extract embeddings
model = SentenceTransformer("all-mpnet-base-v2")
embeddings = model.encode(data, show_progress_bar=True)
```

In [18]:
```
for i in range(3):
    print("ROUND", i)
    params = {
        "embedding_model": "all-mpnet-base-v2",
        "nr_topics": [(i+1)*10 for i in range(5)],
        "min_topic_size": 15,
        "diversity": None,
        "verbose": True
    }

    trainer = Trainer(dataset=dataset,
                      model_name="BERTopic",
                      params=params,
                      bt_embeddings=embeddings,
                      custom_dataset=custom,
                      verbose=True)
    results = trainer.train(save=f"BERTopic_trump_{i+1}")
```

```
ROUND 0

2024-07-25 17:44:33,399 - BERTopic - Reduced dimensionality with UMAP
/opt/conda/lib/python3.10/site-packages/joblib/externals/loky/backend/fork_exec.p
y:38: RuntimeWarning: os.fork() was called. os.fork() is incompatible with multith
readed code, and JAX is multithreaded, so this will likely lead to a deadlock.
  pid = os.fork()
huggingface/tokenizers: The current process just got forked, after parallelism has
already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | f
alse)
huggingface/tokenizers: The current process just got forked, after parallelism has
already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | f
alse)
huggingface/tokenizers: The current process just got forked, after parallelism has
```

Trump - BERTopic (CPU): `all-MiniLM-L6-v2`

In [ ]:
```
%%capture
from sentence_transformers import SentenceTransformer

# Prepare data
dataset, custom = "trump", True
data_loader = DataLoader(dataset)
_, timestamps = data_loader.load_docs()
data = data_loader.load_octis(custom)
data = [" ".join(words) for words in data.get_corpus()]

# Extract embeddings
model = SentenceTransformer("all-MiniLM-L6-v2") # all-MiniLM-L6-v2, all-mpnet-base-v2, u
embeddings = model.encode(data, show_progress_bar=True)
```

In [ ]:
```
%%time
MODEL_NAME = "BERTopic_mini"
for i in range(3):
    print("ROUND", i)
    params = {
        "embedding_model": "all-MiniLM-L6-v2",
        "nr_topics": [(i+1)*10 for i in range(5)],
        "min_topic_size": 15,
        "diversity": None,
        "verbose": True
    }

    trainer = Trainer(dataset=dataset,
                      model_name=MODEL_NAME,
                      params=params,
                      bt_embeddings=embeddings,
                      custom_dataset=custom,
                      verbose=True)
    results = trainer.train(save=f"BERTopic_MiniLM_trump_{i+1}")
```

Trump - BERTopic (CPU): `universal-sentence-encoder` **(USE)**

We modified the `Trainer` class to accommodate the changes required for training with the embeddings `universal-sentence-encoder`

```
In [ ]: class TrainerUSE:

            def __init__(
                self,
                dataset: str,
                model_name: str,
                params: Mapping[str, Any],
                topk: int = 10,
                custom_dataset: bool = False,
                bt_embeddings: np.ndarray = None,
                bt_timestamps: List[str] = None,
                bt_nr_bins: int = None,
                custom_model=None,
                verbose: bool = True,
            ):
                self.dataset = dataset
                self.custom_dataset = custom_dataset
                self.model_name = model_name
                self.params = params
                self.topk = topk
                self.timestamps = bt_timestamps
                self.nr_bins = bt_nr_bins
                self.embeddings = bt_embeddings
                self.ctm_preprocessed_docs = None
                self.custom_model = custom_model
                self.verbose = verbose

                # Prepare data and metrics
                self.data = self.get_dataset()
                self.metrics = self.get_metrics()

                # CTM
                self.qt_ctm = None
                self.training_dataset_ctm = None

            def train(self, save: str = False) -> Mapping[str, Any]:

                results = []

                # Loop over all parameters
                params_name = list(self.params.keys())
                params = {
                    param: (value if type(value) == list else [value])
                    for param, value in self.params.items()
                }
                new_params = list(itertools.product(*params.values()))
                for param_combo in new_params:

                    # Train and evaluate model
                    params_to_use = {
                        param: value for param, value in zip(params_name, param_combo)
                    }
                    output, computation_time = self._train_tm_model(params_to_use)
                    scores = self.evaluate(output)

                    # Update results
                    result = {
```

```
                "Dataset": self.dataset,
                "Dataset Size": len(self.data.get_corpus()),
                "Model": self.model_name,
                "Params": params_to_use,
                "Scores": scores,
                "Computation Time": computation_time,
            }

            result["Params"]["embedding_model"] = "USE"
            results.append(result)

        if save:
            with open(f"{save}.json", "w") as f:
                json.dump(results, f)

            try:
                from google.colab import files

                files.download(f"{save}.json")
            except ImportError:
                pass

        return results

    def _train_tm_model(
        self, params: Mapping[str, Any]
    ) -> Tuple[Mapping[str, Any], float]:
        """Select and train the Topic Model"""
        # Train BERTopic
        if "BERTopic" in self.model_name:
            return self._train_bertopic(params)


    def _train_bertopic(
        self, params: Mapping[str, any]
    ) -> Tuple[Mapping[str, Any], float]:
        """Train BERTopic model"""
        data = self.data.get_corpus()
        data = [" ".join(words) for words in data]
        params["calculate_probabilities"] = False


        if self.custom_model is not None:
            model = self.custom_model(**params)
        else:
            model = BERTopic(**params)

        start = time.time()
        topics, _ = model.fit_transform(data, self.embeddings)


        end = time.time()
        computation_time = float(end - start)

        if not self.timestamps:
            all_words = [word for words in self.data.get_corpus() for word in words]
            bertopic_topics = [
                [
                    vals[0] if vals[0] in all_words else all_words[0]
                    for vals in model.get_topic(i)[:10]
                ]
                for i in range(len(set(topics)) - 1)
            ]

            output_tm = {"topics": bertopic_topics}

        return output_tm, computation_time

    def evaluate(self, output_tm):
        """Using metrics and output of the topic model, evaluate the topic model"""
        if self.timestamps:
            results = {str(timestamp): {} for timestamp, _ in output_tm.items()}
            for timestamp, topics in output_tm.items():
                self.metrics = self.get_metrics()
                for scorers, _ in self.metrics:
                    for scorer, name in scorers:
                        score = scorer.score(topics)
                        results[str(timestamp)][name] = float(score)

        else:
            # Calculate results
            results = {}
            for scorers, _ in self.metrics:
                for scorer, name in scorers:
                    score = scorer.score(output_tm)
                    results[name] = float(score)

            # Print results
            if self.verbose:
                print("Results")
                print("============")
                for metric, score in results.items():
                    print(f"{metric}: {str(score)}")
                print(" ")

        return results

    def get_dataset(self):
        """Get dataset from OCTIS"""
        data = Dataset()

        if self.custom_dataset:
            data.load_custom_dataset_from_folder(self.dataset)
        else:
            data.fetch_dataset(self.dataset)
        return data

    def get_metrics(self):
        """Prepare evaluation measures using OCTIS"""
        npmi = Coherence(texts=self.data.get_corpus(), topk=self.topk, measure="c_npmi"
        topic_diversity = TopicDiversity(topk=self.topk)

        # Define methods
        coherence = [(npmi, "npmi")]
        diversity = [(topic_diversity, "diversity")]
        metrics = [(coherence, "Coherence"), (diversity, "Diversity")]

        return metrics


def sent_to_words(sentences):
    for sentence in sentences:
        yield (gensim.utils.simple_preprocess(str(sentence), deacc=True))
```

In [ ]:
```
%%capture
from sentence_transformers import SentenceTransformer
import tensorflow_hub
import numpy as np
# Prepare data
dataset, custom = "trump", True
data_loader = DataLoader(dataset)
_, timestamps = data_loader.load_docs()
data = data_loader.load_octis(custom)
data = [" ".join(words) for words in data.get_corpus()]

# import tensorflow_hub
model  = tensorflow_hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")
embeddings = model(data)
embeddings = np.array(embeddings)
```

In [ ]:
```
%%time
MODEL_NAME = "BERTopic_USE"
for i in range(3):
    print("ROUND", i)
    params = {
        "embedding_model": model,
        "nr_topics": [(i+1)*10 for i in range(5)],
        "min_topic_size": 15,
        "diversity": None,
        "verbose": True
    }

    trainer = TrainerUSE(dataset=dataset,
                      model_name=MODEL_NAME,
                      params=params,
                      bt_embeddings=embeddings,
                      custom_dataset=custom,
                      verbose=True)
    results = trainer.train(save=f"BERTopic_USE_trump_{i+1}")
```

Data: 20NewsGroup, BBC News

- The code for data processing and model evalations are almost exactly the same as the code for Trump data, so we skip them.

Dynamic topic modeling - BERTopic

- We were only able to run with BERTopic
- Unable to run the LDA Sequence evaluation - too slow to be handled by Kaggle's 12-hour training limit

In [20]:
```
%%capture
from sentence_transformers import SentenceTransformer

# Prepare data
dataset, custom = "trump_dtm", True
data_loader = DataLoader(dataset)
_, timestamps = data_loader.load_docs()
data = data_loader.load_octis(custom)
data = [" ".join(words) for words in data.get_corpus()]

# Extract embeddings
model = SentenceTransformer("all-mpnet-base-v2")
embeddings = model.encode(data, show_progress_bar=True)
```

In [21]:
```
# Match indices
import os
os.listdir(f"./{dataset}")
```

Out[21]:
```
['indexes.txt', 'corpus.tsv', 'vocabulary.txt', 'metadata.json']
```

In [22]:
```
with open(f"./{dataset}/indexes.txt") as f:
    indices = f.readlines()

indices = [int(index.split("\n")[0]) for index in indices]
timestamps = [timestamp for index, timestamp in enumerate(timestamps) if index in indi
len(data), len(timestamps)
```

Out[22]:
```
(44252, 44252)
```

```
In [23]: for i in range(3):
             params = {
                 "nr_topics": [50],
                 "min_topic_size": 15,
                 "verbose": True,
             }

             trainer = Trainer(dataset=dataset,
                               model_name="BERTopic",
                               params=params,
                               bt_embeddings=embeddings,
                               custom_dataset=custom,
                               bt_timestamps=timestamps,
                               topk=5,
                               bt_nr_bins=10,
                               verbose=True)
             results = trainer.train(f"DynamicBERTopic_trump_{i}")
```

```
2024-07-25 18:00:35,412 - BERTopic - Reduced dimensionality with UMAP
2024-07-25 18:00:38,734 - BERTopic - Clustered UMAP embeddings with HDBSCAN
2024-07-25 18:00:44,763 - BERTopic - Reduced number of topics from 362 to 51

0it [00:00, ?it/s]
3it [00:00, 17.27it/s]
5it [00:00,  8.53it/s]
7it [00:00,  8.18it/s]
8it [00:00,  8.49it/s]
9it [00:01,  7.92it/s]
10it [00:01,  7.65it/s]
```

Wall time

- We did not do this part as packages/libraries have changed significantly in the last two years, so the results won't be comparable with the original paper.