07_topic_modelling_falcon

January 21, 2026

1 07_topic_modelling_falcon

This notebook uses an instruction-tuned large language model (Falcon) to extract concise topics from negative Trustpilot reviews, and then applies BERTopic to cluster and summarise those extracted topics. The main steps are:

- Loading the raw filtered review dataset and filtering to negative reviews using rating thresholds defined in the project configuration.
- Optionally sub-sampling the negative reviews for faster experimentation (`SAMPLE_SIZE`), while keeping sampling reproducible via a fixed random seed.
- Running batched topic extraction with the Hugging Face text-generation pipeline using **Falcon-H1-1.5B-Instruct** (`tiiuae/Falcon-H1-1.5B-Instruct`), constrained by:
  - a fixed prompt template
  - maximum input length (`MAX_INPUT_TOKENS`)
  - deterministic decoding (greedy generation with `DO_SAMPLE=False`) for reproducibility
- Preprocessing (using `TextPreproccessor`), parsing and normalising the generated output to extract exactly three short topics per review and filtering out rows with empty or invalid extractions.
- Saving the generated topics and cleaned topic strings as CSV outputs for reporting and downstream modelling.
- Fitting BERTopic (using `BERTopicRunner`) on the cleaned and preprocessed LLM-generated topic strings to identify recurring themes and produce topic-level summaries and visualisations.

The notebook saves outputs to the configured directories:

- Extracted topics and filtered results (CSV): `output/tables/07_topic_modelling_falcon/`
- BERTopic plots (PNG/HTML where applicable): `output/plots/07_topic_modelling_falcon/`
- Saved BERTopic models: `output/models/07_topic_modelling_falcon/`

All experiments were run locally on a MacBook Pro (16-inch, Nov 2024) equipped with an Apple M4 Pro chip (14 CPU cores: 10 performance and 4 efficiency cores) and 24 GB of unified memory, running macOS Tahoe 26.2. Model inference was executed locally; where available, the pipeline can use Apple Silicon acceleration via MPS, otherwise it falls back to CPU. No distributed computing frameworks (e.g. SLURM or OpenPBS) were used. **The notebook has been tested on CPU**

and Apple Silicon (MPS); CUDA support is included for completeness but has not been validated in this project.

The notebook is implemented in Python and relies primarily on pandas, numpy, PyTorch, and Hugging Face Transformers (AutoTokenizer, AutoModelForCausalLM, and pipeline), alongside project-specific utilities for prompt-based topic extraction and BERTopic clustering. For reproducibility, a fixed random seed of **901** was used, and deterministic text generation settings were applied (greedy decoding).

Runtime: On the above hardware, the notebook completes in approximately **4 hours** (MPS) when run on all negative reviews, with runtime dominated by LLM inference (tokenisation and generation). BERTopic clustering on the extracted topics typically adds a smaller additional cost relative to generation. It is recommended **SAMPLE_SIZE = 10** (increase if desired) is set for checking functionality for approximately **1min**.

Reproducibility note (BERTopic plots and top topics)

- BERTopic visualisations in this notebook (for example intertopic distance maps) rely on a UMAP projection. UMAP can yield slightly different embeddings across hardware/OS and execution settings (particularly when parallelism differs), so plots generated on a different device from the one specified above may not exactly match the reported figures even when the same random seed is used. On the device used for this project, the plots are reproducible for the fixed seed (**901**).
- This sensitivity to execution environment is one reason LDA was also explored in this project, as a complementary topic modelling approach with more straightforward reproducibility characteristics.
- Documentation: https://umap-learn.readthedocs.io/en/latest/reproducibility.html
- The reported plots are pre-saved in the folders for this reason. However, executing all notebooks will overwrite them.

[1]:
```
SAMPLE_SIZE = 10  # set None to run all rows

IS_SAMPLE_RUN = SAMPLE_SIZE is not None
SAVE_ARTIFACTS = not IS_SAMPLE_RUN
```

[2]:
```
import time

NOTEBOOK_T0 = time.perf_counter()
print("Notebook timer started.")
```

```
Notebook timer started.
```

[3]:
```
from pathlib import Path
import sys
import ast
from configparser import ConfigParser

# Resolve project root as the parent of the folder the notebook is currently in
```

```
CWD = Path.cwd().resolve()
PROJECT_ROOT = CWD.parent

# Safety fallback
if not (PROJECT_ROOT / "config.ini").exists():
    PROJECT_ROOT = next((p for p in (CWD, *CWD.parents) if (p / "config.ini").
 ↪exists()), None)
    if PROJECT_ROOT is None:
        raise FileNotFoundError("Could not locate 'config.ini' in the current␣
 ↪directory or its parents.")

if str(PROJECT_ROOT) not in sys.path:
    sys.path.insert(0, str(PROJECT_ROOT))

CONFIG = ConfigParser()
CONFIG.read(PROJECT_ROOT / "config.ini")

print("CONFIG used:")
for section in CONFIG.sections():
    print(f"\n[{section}]")
    for key, value in CONFIG[section].items():
        print(f"{key} = {value}")
```

```
CONFIG used:

[DATA]
data_dir = data
raw_filename = PureGym Customer Reviews.csv
raw_filename_filtered = PureGym Customer Reviews_raw_filtered.csv
preprocessed_filename = PureGym Customer Reviews_preprocessed.csv
preprocessed_filename_sentiment = PureGym Customer
Reviews_preprocessed_sentiment.csv
preprocessed_filename_negative = PureGym Customer
Reviews_preprocessed_negative.csv
preprocessed_filename_non_negative = PureGym Customer
Reviews_preprocessed_non_negative.csv
preprocessed_filename_emotion = PureGym Customer
Reviews_preprocessed_emotion.csv
preprocessed_filename_negative_emotion = PureGym Customer
Reviews_preprocessed_negative_emotion.csv
preprocessed_filename_non_negative_emotion = PureGym Customer
Reviews_preprocessed_non_negative_emotion.csv

[OUTPUT]
plot_dir = output/plots
table_dir = output/tables
model_dir = output/models
```

```
[FILTERING]
selected_cols = ["Rating", "Date Experienced", "Review Title", "Review"]
country_code = GB
text_col = Review
detect_language = en
negative_ratings = [1, 2]
emotion_col = Dominant Emotion

[ANALYSIS_DATES]
start_date = 2022-12-17
end_date = 2023-12-17
date_col = Date Experienced

[REPRODUCIBILITY]
seed = 901

[MODELS]
bertopic_negative = bertopic_negative
bertopic_non_negative = bertopic_non_negative
bertopic_emotion_negative_anger = bertopic_emotion_negative_anger
bertopic_emotion_negative_sadness = bertopic_emotion_negative_sad
bertopic_emotion_negative_joy = bertopic_emotion_negative_joy
bertopic_llm_topics_negative = bertopic_llm_topics_negative
bertopic_negative_reviews_llm_filtered = bertopic_negative_reviews_llm_filtered
```

[4]:
```
import pandas as pd
import numpy as np
import random
pd.set_option("display.max_colwidth", None)

from utils.data_management.data_io import load_csv
from modelling.bertopic.bertopic_runner import BERTopicRunner
from utils.processing.text_preprocessor import TextPreprocessor


DATA_DIR = (PROJECT_ROOT / CONFIG["DATA"]["DATA_DIR"])
DATA_DIR.mkdir(parents=True, exist_ok=True)

RAW_FILENAME_FILTERED = CONFIG["DATA"]["RAW_FILENAME_FILTERED"]
RAW_PATH_FILTERED = DATA_DIR / RAW_FILENAME_FILTERED

TEXT_COL = CONFIG["FILTERING"]["TEXT_COL"]
NEGATIVE_RATINGS = ast.literal_eval(CONFIG["FILTERING"].get("NEGATIVE_RATINGS",␣
↪"[]"))
SEED = CONFIG["REPRODUCIBILITY"].getint("SEED")
```

```
random.seed(SEED)
np.random.seed(SEED)

random.seed(SEED)
np.random.seed(SEED)

TABLE_DIR = PROJECT_ROOT / CONFIG["OUTPUT"]["TABLE_DIR"] /␣
  ↪"07_topic_modelling_falcon"
MODEL_DIR = PROJECT_ROOT / CONFIG["OUTPUT"]["MODEL_DIR"] /␣
  ↪"07_topic_modelling_falcon"
PLOT_DIR = PROJECT_ROOT / CONFIG["OUTPUT"]["PLOT_DIR"] /␣
  ↪"07_topic_modelling_falcon"

TABLE_DIR.mkdir(parents=True, exist_ok=True)
MODEL_DIR.mkdir(parents=True, exist_ok=True)
PLOT_DIR.mkdir(parents=True, exist_ok=True)
```

```
/Users/Joshua.Dixon/Documents/8_uni/8 Unstructured Data Analysis/PureGym-NLP-
UDA/.venv/lib/python3.10/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress
not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

[5]:
```
#  Load raw filtered data and filter negative
df_trustpilot = load_csv(str(RAW_PATH_FILTERED))
print("Rows:", len(df_trustpilot))
print("Columns:", df_trustpilot.columns.tolist())

df_negative = df_trustpilot[df_trustpilot["Rating"].isin(NEGATIVE_RATINGS)].
  ↪copy()
print("Negative rows:", len(df_negative))

df_work = df_negative
if SAMPLE_SIZE is not None and len(df_work) > SAMPLE_SIZE:
    df_work = df_work.sample(SAMPLE_SIZE, random_state=SEED)

print("Working rows:", len(df_work))
display(df_work[[TEXT_COL, "Rating"]].head(5))
```

```
Rows: 11300
Columns: ['Rating', 'Date Experienced', 'Review Title', 'Review']
Negative rows: 2368
Working rows: 10

                                                                              ␣
  ↪                                                                            ␣
  ↪                                                                            ␣
  ↪                                          Review  \
11031                                                                          ␣
  ↪                                                                             ␣
  ↪                                                                     your␣
  ↪bring a friend doesn't work well. could be better
3051                    gyms okay, lack of free weights, showers smelt of sewage␣
  ↪and were disgusting to use. cannot find anywhere to report it. could not find␣
  ↪any staff. improve by keeping a better standard of hygiene throughout the gym␣
  ↪and having a reporting feature for improvements
2879   agree with other reviews that pure gym purley has become a filth-pit. its␣
  ↪so dirty - the machines, the changing rooms… ive now started using other␣
  ↪branches which is a shame as this is my closest branch. using the excuse of␣
  ↪reducing paper towel waste is just that - an excuse!
1720                                                                           ␣
  ↪                                                                             ␣
  ↪                          sports water hydration station is not working and␣
  ↪its not for everyone its only for upper package
6967                                                                           ␣
  ↪                                                                             ␣
  ↪                                                         it's to busy, bare␣
  ↪minimum equipment. swim an steam would be better

       Rating
11031       2
3051        2
2879        2
1720        2
6967        1
```

1.1 Falcon Classes

[6]:
```python
from __future__ import annotations

import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline


class FalconInitialiser:
    """Initialise tokenizer, model, and generation pipeline for instruction␣
 ↪models."""

    @staticmethod
    def init_tokenizer(model_name: str) -> AutoTokenizer:
        """Load tokeniser and ensure a pad token is set."""
        tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=True)

        # Setting pad_token avoids warnings and helps with batched generation.␣
 ↪Fall back with eos tokens.
```

```
        if tokenizer.pad_token is None:
            tokenizer.pad_token = tokenizer.eos_token

        return tokenizer

    @staticmethod
    def _has_mps() -> bool:
        """Check whether MPS is built and available."""
        return torch.backends.mps.is_built() and torch.backends.mps.
↪is_available()

    @staticmethod
    def init_model(model_name: str) -> AutoModelForCausalLM:
        """Load model on the best available device."""
        # Prefer CUDA when available
        if torch.cuda.is_available():
            model = AutoModelForCausalLM.from_pretrained(
                model_name,
                dtype=torch.float16,
                device_map="auto",
                low_cpu_mem_usage=True,
            )
            model.eval()
            return model

        # Apple Silicon MPS (next preference))
        if FalconInitialiser._has_mps():
            # Try float16 first for speed/memory, fall back to float32
            try:
                model = AutoModelForCausalLM.from_pretrained(
                    model_name,
                    dtype=torch.float16,
                    low_cpu_mem_usage=True,
                ).to("mps")
            except Exception:
                model = AutoModelForCausalLM.from_pretrained(
                    model_name,
                    dtype=torch.float32,
                    low_cpu_mem_usage=True,
                ).to("mps")

            model.eval()
            return model

        # Final fallback: CPU
        model = AutoModelForCausalLM.from_pretrained(
            model_name,
            low_cpu_mem_usage=True,
        )
        model.eval()
        return model

    @staticmethod
    def init_generator(model_name: str):
        """Create a Transformers text-generation pipeline.

        Returns
        -------
        tuple
            (generator, tokenizer) where generator is a text-generation␣
↪pipeline.
        """

        tokenizer = FalconInitialiser.init_tokenizer(model_name)
        model = FalconInitialiser.init_model(model_name)

        # Choose pipeline device argument based on what's available
        # - CUDA: integer GPU index
        # - MPS: "mps"
        # - CPU: -1
        if torch.cuda.is_available():
            device = 0
        elif FalconInitialiser._has_mps():
            device = "mps"
        else:
            device = -1

        # Build a text-generation pipeline for convenience (handles␣
↪tokenisation + generation)
        generator = pipeline(
            task="text-generation",
            model=model,
            tokenizer=tokenizer,
            return_full_text=False,
            device=device,
        )

        return generator, tokenizer
```

[7]:
```
from dataclasses import dataclass
from typing import Optional


def _prompt_token_len(model_name: str, prompt: str) -> int:
```

```python
    tokenizer = FalconInitialiser.init_tokenizer(model_name)
    return len(tokenizer.encode(prompt, add_special_tokens=False))


@dataclass(frozen=True)
class FalconParams:
    """Configuration for topic extraction via an instruction model."""

    MODEL_NAME: str = "tiiuae/Falcon-H1-1.5B-Instruct"

    # Batching
    BATCH_SIZE: int = 64

    # Generation controls
    MAX_NEW_TOKENS: int = 30 # response tokens,
    DO_SAMPLE: bool = False  # greedy decoding for reproducibility

    # Only used when DO_SAMPLE=True
    TEMPERATURE: Optional[float] = None
    TOP_P: Optional[float] = None

    # Output columns
    RAW_COL: str = "Generated Topics"
    EXTRACTED_COL: str = "Extracted Topics"

    # Prompt
    PROMPT: str = (
        "Extract the 3 main topics from this customer review.\n"
        "Rules:\n"
        "- Return exactly 3 lines and stop.\n"
        "- Each line must be a short noun phrase (max 6 words).\n"
        "- No explanation, no extra text, no follow up questions.\n"
        "Format:\n"
        "1. <topic>\n"
        "2. <topic>\n"
        "3. <topic>\n\n"
        "Review:\n"
    )
    # Keep 512 (see diagnostic in cell below) tokens for the review, plus␣
 ↪however many tokens the prompt uses.
    MAX_INPUT_TOKENS: int = 512 + _prompt_token_len(MODEL_NAME, PROMPT)
```

[8]:
```python
MODEL_NAME = FalconParams.MODEL_NAME
generator, tokenizer = FalconInitialiser.init_generator(MODEL_NAME)
```

```
The fast path is not available because one of `(selective_state_update,
causal_conv1d_fn, causal_conv1d_update)` is None. Falling back to the naive
implementation. To install follow https://github.com/state-
```

```
spaces/mamba/#installation and https://github.com/Dao-AILab/causal-conv1d
Device set to use mps
```

[9]:
```
# Token Length EDA
RUN_TOKEN_LENGTH_DIAGNOSTICS = True
if RUN_TOKEN_LENGTH_DIAGNOSTICS:
    MAXLEN_CANDIDATES = [64, 96, 128, 160, 192, 256, 320, 384, 448, 512]
    df_for_profile = df_negative

    texts = (df_for_profile[TEXT_COL].fillna("").astype(str).str.strip())
    texts = texts.loc[texts.ne("")]

    # Compute token lengths without truncation for each tokeniser
    token_lengths = texts.apply(lambda t: len(tokenizer.encode(t,␣
↪add_special_tokens=True))).to_numpy()

    # Worst-case length per text across both tokenisers (safe choice for later␣
↪max_length)
    token_lengths_worst = token_lengths

    # % fully captured for each candidate max length
    captured_pct = [(token_lengths <= L).mean() * 100 for L in␣
↪MAXLEN_CANDIDATES]
    captured_pct_worst = [(token_lengths_worst <= L).mean() * 100 for L in␣
↪MAXLEN_CANDIDATES]

    # Summary stats
    percentiles = [50, 75, 90, 95, 97, 99]
    pvals = np.percentile(token_lengths, percentiles)


    p99 = float(np.percentile(token_lengths, 99))

    print("Token length percentiles:")
    for p, v in zip(percentiles, pvals):
        print(f"{p:>2}%: {int(v)} tokens")


    print(f"\n99th percentile: {int(p99)} tokens")

    best_L = next((L for L, p in zip(MAXLEN_CANDIDATES, captured_pct_worst) if␣
↪p >= 99), None)
```

```
Token length percentiles:
50%: 61 tokens
75%: 114 tokens
90%: 193 tokens
95%: 269 tokens
```

```
97%: 315 tokens
99%: 529 tokens

99th percentile: 529 tokens
```

[10]:
```
print("MAX_INPUT_TOKENS:", FalconParams.MAX_INPUT_TOKENS)
```

```
MAX_INPUT_TOKENS: 595
```

[11]:
```
from __future__ import annotations

from dataclasses import dataclass
from typing import Any, Optional

import pandas as pd
import re
import torch


@dataclass(frozen=True)
class FalconBatchResult:
    """Container for batched generation results."""
    # Raw decoded text returned by the generation pipeline (one per input row)
    raw_outputs: list[str]
    # Parsed topics extracted from the raw output (one per input row)
    extracted_topics: list[str]


class FalconDataCompiler:
    """Apply an instruction model to extract 3 topics per row."""

    def __init__(
        self,
        *,
        df: pd.DataFrame,
        text_col: str,
        generator: Any,
        tokenizer: Any,
        prompt_template: str,
        batch_size: int,
        max_input_tokens: int,
        max_new_tokens: int,
        temperature: Optional[float],
        top_p: Optional[float],
        do_sample: bool,
        raw_output_col: str,
        topics_col: str,
    ) -> None:
        if text_col not in df.columns:
            raise KeyError(f"'{text_col}' not found in DataFrame")

        # Store core inputs
        self._df = df
        self._text_col = text_col

        # Hugging Face text-generation pipeline and matching tokenizer
        self._generator = generator
        self._tokenizer = tokenizer

        # Prompt template and batching controls
        self._prompt_template = str(prompt_template)
        self._batch_size = int(batch_size)

        # Token budgets
        # max_input_tokens controls how many tokens we allow in the full␣
↪prompt+review
        # max_new_tokens controls how many tokens the model is allowed to␣
↪generate as output
        self._max_input_tokens = int(max_input_tokens)
        self._max_new_tokens = int(max_new_tokens)

        # Whether to sample (stochastic decoding) or use greedy decoding␣
↪(deterministic)
        self._do_sample = bool(do_sample)

        # Only keep sampling params when sampling is enabled
        self._temperature: Optional[float] = float(temperature) if (self.
↪_do_sample and temperature is not None) else None
        self._top_p: Optional[float] = float(top_p) if (self._do_sample and␣
↪top_p is not None) else None

        # Output column names
        self._raw_output_col = str(raw_output_col)
        self._topics_col = str(topics_col)

        # Support both templates that include "{review}" and templates that do␣
↪not (experimented both cases in development)
        # If "{review}" exists, we split into prefix + suffix so we can␣
↪token-budget just the review content
        if "{review}" in self._prompt_template:
            prefix, suffix = self._prompt_template.split("{review}", 1)
            self._prompt_prefix = prefix
            self._prompt_suffix = suffix
        else:
            self._prompt_prefix = self._prompt_template
            self._prompt_suffix = ""

        # Pre-tokenise the prefix and suffix once for efficiency and stable␣
↪budgeting
        self._prefix_ids = self._tokenizer.encode(self._prompt_prefix,␣
↪add_special_tokens=False)
        self._suffix_ids = self._tokenizer.encode(self._prompt_suffix,␣
↪add_special_tokens=False)

        # Ensure the prompt scaffold alone does not exceed the model input␣
↪budget
        reserved = len(self._prefix_ids) + len(self._suffix_ids)
        if reserved >= self._max_input_tokens:
            raise ValueError(
                "Prompt is longer than MAX_INPUT_TOKENS. Reduce␣
↪MAX_INPUT_TOKENS or shorten PROMPT."
            )

    def apply(self) -> pd.DataFrame:
        """Run generation for all rows and return a copy with new columns."""
        df_out = self._df.copy()

        # Pull the text column into a plain list
        texts = (
            df_out[self._text_col]
            .fillna("")
            .astype(str)
            .tolist()
        )

        # Run batched generation + parsing
        batch_result = self._predict(texts)

        # Append outputs back onto the DataFrame
        df_out[self._raw_output_col] = batch_result.raw_outputs
        df_out[self._topics_col] = batch_result.extracted_topics

        return df_out

    def _predict(self, texts: list[str]) -> FalconBatchResult:
        """Generate in batches and extract numbered topics."""
        # Handle empty inputs
        if not texts:
            return FalconBatchResult(raw_outputs=[], extracted_topics=[])

        # Pre-allocate output lists
        raw_outputs: list[str] = [""] * len(texts)
        extracted_topics: list[str] = [""] * len(texts)

        # Determine pad token for generation; fall back to EOS if pad is␣
↪unavailable
        pad_id = getattr(self._tokenizer, "pad_token_id", None)
        if pad_id is None:
            pad_id = getattr(self._tokenizer, "eos_token_id", None)

        # Loop over input texts in batches for speed/memory control
        for start in range(0, len(texts), self._batch_size):
            batch = texts[start: start + self._batch_size]

            # Build prompts with per-review truncation to respect␣
↪max_input_tokens
            prompts = [self._build_prompt(t) for t in batch]

            # Base generation kwargs shared for greedy and sampling modes
            gen_kwargs: dict[str, Any] = dict(
                max_new_tokens=self._max_new_tokens,
                do_sample=self._do_sample,
                pad_token_id=pad_id,
            )

            # Add sampling-specific kwargs only when enabled
            if self._do_sample:
                if self._temperature is not None:
                    gen_kwargs["temperature"] = self._temperature
                if self._top_p is not None:
                    gen_kwargs["top_p"] = self._top_p

            # Inference only: no gradients needed
            with torch.inference_mode():
                outputs = self._generator(prompts, **gen_kwargs)

            # Normalise the pipeline outputs and extract three numbered topics
            for i, out in enumerate(outputs):
                idx = start + i
                generated = self._normalise_pipeline_output(out)
                raw_outputs[idx] = generated

                topics = self._extract_numbered_topics(generated, n=3)
                extracted_topics[idx] = " | ".join(topics) if topics else ""

        return FalconBatchResult(raw_outputs=raw_outputs,␣
↪extracted_topics=extracted_topics)
```

```
    def _build_prompt(self, review_text: str) -> str:
        """Construct a prompt with review text truncated to fit␣
↪max_input_tokens."""
        review = str(review_text)

        # Compute the token budget available for the review after accounting␣
↪for prefix + suffix
        budget = self._max_input_tokens - (len(self._prefix_ids) + len(self.
↪_suffix_ids))

        # Tokenise the review without special tokens so the budget aligns with␣
↪prefix/suffix tokenisation
        review_ids = self._tokenizer.encode(review, add_special_tokens=False)

        # Truncate review tokens if needed to ensure the full prompt fits␣
↪within max_input_tokens
        if len(review_ids) > budget:
            review_ids = review_ids[:budget]

        # Combine and decode back to text for the generation pipeline
        full_ids = self._prefix_ids + review_ids + self._suffix_ids
        return self._tokenizer.decode(full_ids, skip_special_tokens=True)

    @staticmethod
    def _normalise_pipeline_output(item: Any) -> str:
        """Normalise pipeline outputs to a single generated text string."""
        # HF text-generation pipeline returns:
        # - list[dict] with key "generated_text"
        # - dict with key "generated_text"
        # - other string-like structures depending on wrapper/version
        if isinstance(item, list) and item:
            item0 = item[0]
            if isinstance(item0, dict) and "generated_text" in item0:
                return str(item0["generated_text"])
            return str(item0)

        if isinstance(item, dict) and "generated_text" in item:
            return str(item["generated_text"])

        return str(item)

    @staticmethod
    def _extract_numbered_topics(text: str, *, n: int) -> list[str]:
        """Extract up to n topics from numbered output (1., 2., 3.)."""
        s = str(text)

        topics: list[str] = []
        # Preferred: topics appear as separate numbered lines
        for line in s.splitlines():
            match = re.match(r"^\s*([1-9])[\.\)]\s*(.+?)\s*$", line)
            if match:
                topics.append(match.group(2).strip())
            if len(topics) >= n:
                return topics[:n]

        # Fallback: topics might be inline in a single block of text
        inline: list[str] = []
        for m in re.finditer(r"([1-9])[\.\)]\s*(.+?)(?=\s+[1-9][\.\)]\s*|$)",␣
↪s):
            inline.append(m.group(2).strip())
            if len(inline) >= n:
                break

        return inline[:n]
```

[12]:
```
compiler = FalconDataCompiler(
    df=df_work,
    text_col=TEXT_COL,
    generator=generator,
    tokenizer=tokenizer,
    prompt_template=FalconParams.PROMPT,
    batch_size=FalconParams.BATCH_SIZE,
    max_input_tokens=FalconParams.MAX_INPUT_TOKENS,
    max_new_tokens=FalconParams.MAX_NEW_TOKENS,
    temperature=FalconParams.TEMPERATURE,
    top_p=FalconParams.TOP_P,
    do_sample=FalconParams.DO_SAMPLE,
    raw_output_col=FalconParams.RAW_COL,
    topics_col=FalconParams.EXTRACTED_COL,
)

df_llm = compiler.apply()
display(df_llm[[TEXT_COL, FalconParams.RAW_COL, FalconParams.EXTRACTED_COL]].
↪head(5))

if SAVE_ARTIFACTS:
    out_all = TABLE_DIR / "llm_topics_negative_processed.csv"
    df_llm.to_csv(out_all, index=False)
    print("Saved:", out_all)
else:
    print("Sample run: skipping save of llm_topics_negative_processed.csv")
```

```python
df_llm_filtered = df_llm[df_llm[FalconParams.EXTRACTED_COL].astype(str).str.
 ↪strip().ne("")].copy()
print("Rows with extracted topics:", len(df_llm_filtered))

if SAVE_ARTIFACTS:
    out_filtered = TABLE_DIR / "llm_topics_negative_processed_filtered.csv"
    df_llm_filtered.to_csv(out_filtered, index=False)
    print("Saved:", out_filtered)
else:
    print("Sample run: skipping save of llm_topics_negative_processed_filtered.
 ↪csv")

display(df_llm_filtered[[TEXT_COL, FalconParams.EXTRACTED_COL]].head(5))

print("Tokenizer class:", type(tokenizer).__name__)
print("Is fast tokenizer:", getattr(tokenizer, "is_fast", False))
print("Vocab size:", tokenizer.vocab_size)
print("Model max length:", tokenizer.model_max_length)
print("Pad token:", tokenizer.pad_token, tokenizer.pad_token_id)
print("EOS token:", tokenizer.eos_token, tokenizer.eos_token_id)
```

```
The following generation flags are not valid and may be ignored:
['temperature']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

                                                                               ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                   Review  \
11031                                                                          ␣
 ↪                                                                              ␣
 ↪                                                                       your␣
 ↪bring a friend doesn’t work well. could be better
3051                    gyms okay, lack of free weights, showers smelt of sewage␣
 ↪and were disgusting to use. cannot find anywhere to report it. could not find␣
 ↪any staff. improve by keeping a better standard of hygiene throughout the gym␣
 ↪and having a reporting feature for improvements
2879   agree with other reviews that pure gym purley has become a filth-pit. its␣
 ↪so dirty - the machines, the changing rooms… ive now started using other␣
 ↪branches which is a shame as this is my closest branch. using the excuse of␣
 ↪reducing paper towel waste is just that - an excuse!
1720                                                                           ␣
 ↪                                                                              ␣
 ↪                           sports water hydration station is not working and␣
 ↪its not for everyone its only for upper package
6967                                                                           ␣
 ↪                                                                              ␣
 ↪                                                            it’s to busy, bare␣
 ↪minimum equipment. swim an steam would be better
```

```
                                                                              ␣
 ↪                                             Generated Topics  \
11031                 . i bought it for my 12 year old daughter and she loves it.
 ↪ she is very creative and loves to draw. she loves the idea
3051                                                          . \n\n1. Lack␣
 ↪of free weights.\n2. Smelly showers.\n3. Reporting feature.
2879          - the staff are also not very helpful. ive tried to contact them␣
 ↪via email and phone, but so far i have not received any reply
1720                  . i bought it for my 12 year old daughter who is very␣
 ↪active and loves sports. she is very picky about her drink and i
6967   . the pool is small and the water is cold. the staff are very friendly␣
 ↪but the equipment is outdated. i would recommend this place to

                                                   Extracted Topics
11031
3051   Lack of free weights. | Smelly showers. | Reporting feature.
2879
1720
6967

Sample run: skipping save of llm_topics_negative_processed.csv
Rows with extracted topics: 2
Sample run: skipping save of llm_topics_negative_processed_filtered.csv

                                                                              ␣
 ↪                                                                             ␣
 ↪                                                                             ␣
 ↪                       Review  \
3051  gyms okay, lack of free weights, showers smelt of sewage and were␣
 ↪disgusting to use. cannot find anywhere to report it. could not find any staff.
 ↪ improve by keeping a better standard of hygiene throughout the gym and having␣
 ↪a reporting feature for improvements
9089                                                                          ␣
 ↪                    the lighting is so poor and demotivating. plus there's never␣
 ↪staff in the gym to ask any equipment queries. alongside, the shower would␣
 ↪clog whenever you did use them.

                                                  Extracted Topics
3051  Lack of free weights. | Smelly showers. | Reporting feature.
9089                                                 lighting is poor.

Tokenizer class: PreTrainedTokenizerFast
Is fast tokenizer: True
Vocab size: 65536
Model max length: 1000000000000000019884624838656
Pad token: <pad> 65536
EOS token: <|end_of_text|> 11
```

```
[13]: # Re-load from disk to ensure downstream steps always use saved artefacts
TOPICS_PATH = PROJECT_ROOT / CONFIG["OUTPUT"]["TABLE_DIR"] /␣
 ↪"07_topic_modelling_falcon" / "llm_topics_negative_processed.csv"
TOPICS_PATH_FILTERED = PROJECT_ROOT / CONFIG["OUTPUT"]["TABLE_DIR"] /␣
 ↪"07_topic_modelling_falcon" / "llm_topics_negative_processed_filtered.csv"

df_llm_reloaded = load_csv(TOPICS_PATH)
df_llm_filtered_reloaded = load_csv(TOPICS_PATH_FILTERED)

df_topics_for_bertopic = df_llm_filtered_reloaded.copy()

# Prepare extracted topics for BERTopic
def normalise_llm_topics(text: str) -> str:
    """Normalise extracted topics into a single whitespace-separated string."""
    if not isinstance(text, str):
        return ""
    t = text.strip()
    if not t:
        return ""

    # If any numbering slipped in, remove it
    t = re.sub(r"(?m)^\s*\d+\s*[\.\)\-:]\s*", "", t)

    # Convert pipe-separated topics to a single string
    t = t.replace("|", " ")

    # Normalise whitespace
    t = re.sub(r"\s+", " ", t).strip()
    return t

# This is the Falcon-derived text column for modelling
FALCON_TOPIC_TEXT_COL = "LLM Topics Clean"
df_topics_for_bertopic[FALCON_TOPIC_TEXT_COL] =␣
 ↪df_topics_for_bertopic[FalconParams.EXTRACTED_COL].
 ↪apply(normalise_llm_topics)
df_topics_for_bertopic =␣
 ↪df_topics_for_bertopic[df_topics_for_bertopic[FALCON_TOPIC_TEXT_COL].ne("")]

print("Rows going into BERTopic (topics):", len(df_topics_for_bertopic))

STOPWORD_LANGUAGE = "english"
EXTRA_STOPWORDS = ["pure", "gym", "puregym", # "equipment" retained (removing␣
 ↪it reduced topic stability and increased fragmentation for this modelling)
                   "main", "question", "extracted", "topics", "review",␣
 ↪"answer" # Additional stopwords associated with generated responses from␣
 ↪model.
                   ]
```

```
PUNCTUATION_PATTERN_TOPIC = r"[-.,\"'’`;:!?()/&%]+"
USE_POS_TAGGING_TOPIC = False

df_topics_for_bertopic[FALCON_TOPIC_TEXT_COL] = TextPreprocessor(
    punctuation_pattern=PUNCTUATION_PATTERN_TOPIC,
    extra_stopwords=EXTRA_STOPWORDS,
    use_pos_tagging=USE_POS_TAGGING_TOPIC,
    language=STOPWORD_LANGUAGE,
).transform_many(df_topics_for_bertopic[FALCON_TOPIC_TEXT_COL])

df_topics_for_bertopic =␣
 ↪df_topics_for_bertopic[df_topics_for_bertopic[FALCON_TOPIC_TEXT_COL].
 ↪astype(str).str.strip().ne("")]

display(df_topics_for_bertopic[[FalconParams.EXTRACTED_COL,␣
 ↪FALCON_TOPIC_TEXT_COL]].head(5))
```

```
Rows going into BERTopic (topics): 1087

                                                        Extracted Topics  \
0  upkeep of machines/facilities | mens showers | membership fee increase
1      Joining fee issue. | Gym closure issue. | Conversion policy issue.
2                                 gym shut for almost a week. | i joined
3                              bad equipment | no staff | bad management
4                          gym size. | facilities issues. | reopen date.

                                         LLM Topics Clean
0  upkeep machine facility men shower membership fee increase
1     joining fee issue closure issue conversion policy issue
2                                      shut almost week joined
3                         bad equipment staff bad management
4                               size facility issue reopen date
```

[14]:
```
# Note: SAMPLE_SIZE only affects Falcon extraction. BERTopic is always run on␣
 ↪the imported (saved) full extracted topic dataset.
#       This approach has been adopted as BERTopic struggles to find topics on␣
 ↪small samples and creates errors when running.
TOP_N_TOPICS = 4
N_WORDS_BARCHART = 5
LABEL = "llm_topics_negative"

UMAP_N_NEIGHBOURS = 15
UMAP_N_COMPONENTS = 5
UMAP_MIN_DIST = 0
UMAP_METRIC = "cosine"

runner = BERTopicRunner(
    model_dir=MODEL_DIR,
    plot_dir=PLOT_DIR,
    table_dir=TABLE_DIR,
    seed=SEED,
    top_n_topics=TOP_N_TOPICS,
    n_words_barchart=N_WORDS_BARCHART,
    min_topic_size=30,
    show_plots=True,
    save_png=True,
    png_scale=2,


    # UMAP controls
    umap_n_neighbors=UMAP_N_NEIGHBOURS,
    umap_n_components=UMAP_N_COMPONENTS,
    umap_min_dist=UMAP_MIN_DIST,
    umap_metric=UMAP_METRIC,
)

result_llm_topics = runner.run(
    df_topics_for_bertopic,
    label=LABEL,
    text_col=FALCON_TOPIC_TEXT_COL,
    verbose=True,
)

print(result_llm_topics.plot_paths)
display(result_llm_topics.topic_info.head(5))
display(result_llm_topics.top_topics_table.head(5))
```

```
2026-01-21 17:34:30,423 - BERTopic - Embedding - Transforming documents to
embeddings.
Batches: 100%|      | 34/34 [00:00<00:00, 67.18it/s]
2026-01-21 17:34:33,129 - BERTopic - Embedding - Completed
2026-01-21 17:34:33,129 - BERTopic - Dimensionality - Fitting the dimensionality
reduction algorithm
2026-01-21 17:34:37,296 - BERTopic - Dimensionality - Completed
2026-01-21 17:34:37,297 - BERTopic - Cluster - Start clustering the reduced
embeddings
2026-01-21 17:34:37,310 - BERTopic - Cluster - Completed
2026-01-21 17:34:37,312 - BERTopic - Representation - Fine-tuning topics using
representation models.
2026-01-21 17:34:37,320 - BERTopic - Representation - Completed
2026-01-21 17:34:37,330 - BERTopic - WARNING: When you use `pickle` to save/load
a BERTopic model,please make sure that the environments in which you saveand
load the model are **exactly** the same. The version of BERTopic,its
dependencies, and python need to remain the same.
```

```
{'intertopic_distance_html': PosixPath('/Users/Joshua.Dixon/Documents/8_uni/8
Unstructured Data Analysis/PureGym-NLP-UDA/output/plots/07_topic_modelling_falco
n/bertopic_llm_topics_negative_intertopic_distance.html'),
'intertopic_distance_png': PosixPath('/Users/Joshua.Dixon/Documents/8_uni/8
Unstructured Data Analysis/PureGym-NLP-UDA/output/plots/07_topic_modelling_falco
n/bertopic_llm_topics_negative_intertopic_distance.png'),
'barchart_top_topics_html': PosixPath('/Users/Joshua.Dixon/Documents/8_uni/8
Unstructured Data Analysis/PureGym-NLP-UDA/output/plots/07_topic_modelling_falco
n/bertopic_llm_topics_negative_barchart_top4.html'), 'barchart_top_topics_png':
PosixPath('/Users/Joshua.Dixon/Documents/8_uni/8 Unstructured Data
Analysis/PureGym-NLP-UDA/output/plots/07_topic_modelling_falcon/bertopic_llm_top
ics_negative_barchart_top4.png'), 'heatmap_all_topics_html':
PosixPath('/Users/Joshua.Dixon/Documents/8_uni/8 Unstructured Data
Analysis/PureGym-NLP-UDA/output/plots/07_topic_modelling_falcon/bertopic_llm_top
ics_negative_heatmap_all_topics.html'), 'heatmap_all_topics_png':
PosixPath('/Users/Joshua.Dixon/Documents/8_uni/8 Unstructured Data
Analysis/PureGym-NLP-UDA/output/plots/07_topic_modelling_falcon/bertopic_llm_top
ics_negative_heatmap_all_topics.png'), 'heatmap_top_topics_html':
PosixPath('/Users/Joshua.Dixon/Documents/8_uni/8 Unstructured Data
Analysis/PureGym-NLP-UDA/output/plots/07_topic_modelling_falcon/bertopic_llm_top
ics_negative_heatmap_top4.html'), 'heatmap_top_topics_png':
PosixPath('/Users/Joshua.Dixon/Documents/8_uni/8 Unstructured Data
Analysis/PureGym-NLP-UDA/output/plots/07_topic_modelling_falcon/bertopic_llm_top
ics_negative_heatmap_top4.png'), 'topic_info_csv':
PosixPath('/Users/Joshua.Dixon/Documents/8_uni/8 Unstructured Data
Analysis/PureGym-NLP-UDA/output/tables/07_topic_modelling_falcon/bertopic_llm_to
pics_negative_topic_info.csv'), 'top_topics_csv':
PosixPath('/Users/Joshua.Dixon/Documents/8_uni/8 Unstructured Data
Analysis/PureGym-NLP-UDA/output/tables/07_topic_modelling_falcon/bertopic_llm_to
pics_negative_top_topics.csv')}

   Topic  Count                                 Name  \
0     -1    342  -1_machine_equipment_parking_broken
1      0    248    0_membership_customer_service_fee
2      1     85             1_shower_water_cold_room
3      2     81           2_toilet_smell_smelly_room
4      3     79       3_equipment_space_staff_weight

                                                                               ␣
 ↪Representation  \
0        [machine, equipment, parking, broken, staff, bad, issue, topic, room,␣
 ↪trainer]
1  [membership, customer, service, fee, bad, issue, cancellation, email, access,␣
 ↪price]
2           [shower, water, cold, room, locker, hot, temperature, need, warm,␣
 ↪facility]
3  [toilet, smell, smelly, room, changing, dirty, broken, urine, management,␣
 ↪equipment]
```

```
4     [equipment, space, staff, weight, cardio, machine, bad, treadmill, broken,␣
↪bench]

                                                                                ␣
↪                                                           Representative_Docs
0  [equipment broken changing room dirty need, bad music broken equipment poor␣
↪customer service, broken machine poor customer service broken locker topic]
1                                                                  [bad customer␣
↪service, bad customer service, customer service bad membership cancelled]
2                                                                               ␣
↪                            [shower cold staff, shower cold, shower bad]
3                  [changing room dirty smell bad cleaning poor, tragic changing␣
↪room smell toilet, aircon working smell lady changing room toilet working]
4                                          [space good maintenance bad broken␣
↪equipment, messy cardio machine space weight area, broken equipment bad staff]
```

```
   Topic  Count  \
0      0    248
1      1     85
2      2     81
3      3     79

                                                                            ␣
↪TopWords
0  membership, customer, service, fee, bad, issue, cancellation, email, access,␣
↪price
1          shower, water, cold, room, locker, hot, temperature, need, warm,␣
↪facility
2  toilet, smell, smelly, room, changing, dirty, broken, urine, management,␣
↪equipment
3     equipment, space, staff, weight, cardio, machine, bad, treadmill, broken,␣
↪bench
```

[ ]:
```
elapsed_s = time.perf_counter() - NOTEBOOK_T0
elapsed_m = elapsed_s / 60
print(f"\nTotal runtime: {elapsed_s:,.1f} seconds ({elapsed_m:,.2f} minutes)")
```

```
Total runtime: 66.5 seconds (1.11 minutes)
```