3_custom_dataset_creation

December 27, 2025

1 Migration dataset

This notebook contains the code I used to compile my own custom dataset on the topic of migration. The texts that can be produced with this code need some post-cleaning, like putting them into the same file, checking if all characters are rendered correctly etc..

2 Human texts

The human texts were taken from the SOCC corpus as published in the file SOCC_gnm_articles.csv, which can be downloaded as part of the SOCC zip from here: https://github.com/sfu-discourse-lab/SOCC/releases

- First, the 10'366 articles can be searched for migration terms and only the 466 of them containing at least 2 migration terms will be kept.
- Secondly, a zero-shot topic classification step will be applied using a BART-based natural language inference model, classifying texts into four coarse topical categories, including a migration-specific label.

[ ]:
```
import pandas as pd
import re

# --- Config ---
input_path  = "C:\\Users\\andre\\OneDrive\\Desktop\\PAN Task␣
↪2025\\SOCC_data\\SOCC_gnm_articles.csv"
filtered_texts = "C:\\Users\\andre\\OneDrive\\Desktop\\PAN Task␣
↪2025\\SOCC_data\\SOCC_gnm_articles_migration.csv"

migration_terms = [
    "immigration", "immigrant", "migrant", "migration",
    "refugee", "asylum", "border", "deportation",
    "undocumented", "illegal", "visa", "integration"
]

# Compile regex patterns (word-boundary aware, case-insensitive)
term_patterns = {
    term: re.compile(rf"\b{re.escape(term)}\b", re.IGNORECASE)
    for term in migration_terms
}
```

```
# --- Load data ---
df = pd.read_csv(input_path)

# --- Function to count distinct matched terms ---
def find_migration_terms(text):
    if pd.isna(text):
        return []
    return [
        term for term, pattern in term_patterns.items()
        if pattern.search(text)
    ]

# Apply matching
df["matched_terms"] = df["article_text"].apply(find_migration_terms)
df["n_matched_terms"] = df["matched_terms"].apply(len)

# Filter: at least 2 distinct migration terms
df_filtered = df[df["n_matched_terms"] >= 2].copy()

# --- Save ---
df_filtered.to_csv(filtered_texts, index=False)

print(f"Saved {len(df_filtered)} articles to {filtered_texts}")
```

[ ]:
```
from transformers import pipeline
import pandas as pd
import numpy as np
from tqdm import tqdm

tqdm.pandas()

classifier = pipeline(
    "zero-shot-classification",
    model="facebook/bart-large-mnli",
    device=0
)

candidate_labels = [
    "This article is mainly about immigration or migration policy",
    "This article is mainly about economics or fiscal policy",
    "This article is mainly about culture, identity, or values",
    "This article is mainly about other political or social issues"
]

import re
```

```
def clean_html(text):
    text = re.sub(r"<.*?>", " ", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip()

def classify_article(text):
    cleaned = clean_html(text)

    result = classifier(
        cleaned[:4000],  # safety cap; BART max is 1024 tokens
        candidate_labels,
        multi_label=False
    )

    return pd.Series({
        "zs_top_label": result["labels"][0],
        "zs_top_score": result["scores"][0],
        "zs_second_score": result["scores"][1]
    })

df = pd.read_csv(filtered_texts)

zs_results = df["article_text"].progress_apply(classify_article)

df = pd.concat([df, zs_results], axis=1)

MIGRATION_LABEL = "This article is mainly about immigration or migration policy"

df_core = df[
    (df["zs_top_label"] == MIGRATION_LABEL) &
    (df["zs_top_score"] >= 0.6)
]

df_peripheral = df[
    df["article_text"].str.contains(
        r"\b(immigration|immigrant|migrant|refugee|asylum)\b",
        case=False,
        regex=True
    ) &
    ~df.index.isin(df_core.index)
]
```

[ ]:
```
print("Total articles:", len(df))
print("Core migration articles:", len(df_core))
print("Peripheral migration articles:", len(df_peripheral))
```

3 AI generated texts

The AI generated texts were generated by sending one of three prompts at random to openai, gemini or deepseek. To run this code, you will have to enter your own API key.

```
[ ]: import openai
     import random
     import re
     import os
     import csv
     from tqdm import tqdm

     os.environ["OPENAI_API_KEY"] = #enter your API key
     openai.api_key = os.environ["OPENAI_API_KEY"]

     client = OpenAI()

     PROMPTS = [
         "Write a newspaper opinion piece about immigration.",
         "Write an editorial on immigration for a general newspaper audience.",
         "Write a commentary article discussing immigration policy and society."
     ]

     def trim_to_last_sentence(text, max_len):
         text = text[:max_len]
         matches = list(re.finditer(r'[.!?]["\']?\s', text))
         if matches:
             return text[:matches[-1].end()].strip()
         return text.strip()

     def generate_text(min_len, max_len, model="gpt-4o-mini"):
         target_length = random.randint(min_len, max_len)
         base_prompt = random.choice(PROMPTS)

         prompt = (
             f"{base_prompt} "
             f"Suitable for publication in a general-interest newspaper. "
             f"Length: approximately {target_length} characters."
         )

         response = client.chat.completions.create(
             model=model,
             messages=[
                 {"role": "system", "content": "You are a professional newspaper␣
       ↪columnist."},
                 {"role": "user", "content": prompt}
             ],
             temperature=0.8
         )

         text = response.choices[0].message.content
         text = trim_to_last_sentence(text, target_length)

         return {
             "text": text,
             "target_length": target_length,
             "actual_length": len(text),
             "model": model
         }

     samples = []

     distributions = [
         (25, 1500, 3500),
         (50, 3500, 5500),
         (25, 5500, 7500)
     ]

     for count, min_len, max_len in distributions:
         for _ in tqdm(range(count), desc=f"{min_len}-{max_len} chars"):
             sample = generate_text(min_len, max_len)
             samples.append(sample)

             output_file = "gpt_immigration_texts.csv"

     with open(output_file, mode="w", newline="", encoding="utf-8") as f:
         writer = csv.DictWriter(
             f,
             fieldnames=["model", "target_length", "actual_length", "text"]
         )
         writer.writeheader()
         writer.writerows(samples)

     print(f"Saved {len(samples)} texts to {output_file}")
```

[ ]:
```
import random
import re
import os
import csv
from tqdm import tqdm
from openai import OpenAI
# ---- DeepSeek API setup ----
os.environ["DEEPSEEK_API_KEY"] = #enter your API key
```

```python
client = OpenAI(
    api_key=os.environ["DEEPSEEK_API_KEY"],
    base_url="https://api.deepseek.com"
)

# ---- Prompts ----
PROMPTS = [
    "Write a newspaper opinion piece about immigration.",
    "Write an editorial on immigration for a general newspaper audience.",
    "Write a commentary article discussing immigration policy and society."
]

# ---- Trimming ----
def trim_to_last_sentence(text, max_len):
    text = text[:max_len]
    matches = list(re.finditer(r'[.!?]["\']?\s', text))
    if matches:
        return text[:matches[-1].end()].strip()
    return text.strip()

# ---- Generation ----
def generate_text(min_len, max_len, model="deepseek-chat"):
    target_length = random.randint(min_len, max_len)
    base_prompt = random.choice(PROMPTS)

    prompt = (
        f"{base_prompt} "
        f"Suitable for publication in a general-interest newspaper. "
        f"Length: approximately {target_length} characters."
    )

    response = client.chat.completions.create(
        model=model,
        messages=[
            {"role": "system", "content": "You are a professional newspaper␣
 ↪columnist."},
            {"role": "user", "content": prompt}
        ],
        temperature=0.8
    )

    text = response.choices[0].message.content
    text = trim_to_last_sentence(text, target_length)

    return {
        "text": text,
        "target_length": target_length,
        "actual_length": len(text),
        "model": model
    }

# ---- Sampling ----
samples = []

distributions = [
    (25, 1500, 3500),
    (50, 3500, 5500),
    (25, 5500, 7500)
]

for count, min_len, max_len in distributions:
    for _ in tqdm(range(count), desc=f"{min_len}-{max_len} chars"):
        samples.append(generate_text(min_len, max_len))

# ---- Save ----
output_file = "deepseek_immigration_texts.csv"

with open(output_file, mode="w", newline="", encoding="utf-8") as f:
    writer = csv.DictWriter(
        f,
        fieldnames=["model", "target_length", "actual_length", "text"]
    )
    writer.writeheader()
    writer.writerows(samples)

print(f"Saved {len(samples)} texts to {output_file}")
```

[ ]:
```
import random
import re
import os
import csv
from tqdm import tqdm
import google.generativeai as genai

# ---- Gemini API setup ----
os.environ["GEMINI_API_KEY"] = #enter your API key
genai.configure(api_key=os.environ["GEMINI_API_KEY"])

model = genai.GenerativeModel("models/gemini-flash-lite-latest")

# ---- Prompts ----
PROMPTS = [
    "Write a newspaper opinion piece about immigration.",
    "Write an editorial on immigration for a general newspaper audience.",
    "Write a commentary article discussing immigration policy and society."
]

# ---- Trimming ----
def trim_to_last_sentence(text, max_len):
    text = text[:max_len]
    matches = list(re.finditer(r'[.!?]["\']?\s', text))
    if matches:
        return text[:matches[-1].end()].strip()
    return text.strip()

# ---- Generation ----
def generate_text(min_len, max_len):
    target_length = random.randint(min_len, max_len)
    base_prompt = random.choice(PROMPTS)

    prompt = (
        f"{base_prompt} "
        f"Suitable for publication in a general-interest newspaper. "
        f"Length: approximately {target_length} characters."
    )

    response = model.generate_content(
        prompt,
        generation_config={
            "temperature": 0.8
        }
    )

    text = response.text
    text = trim_to_last_sentence(text, target_length)

    return {
        "text": text,
        "target_length": target_length,
        "actual_length": len(text),
        "model": "gemini-pro"
    }

# ---- Sampling ----
samples = []

distributions = [
    (25, 1500, 3500),
    (50, 3500, 5500),
    (25, 5500, 7500)
]
```

```python
for count, min_len, max_len in distributions:
    for _ in tqdm(range(count), desc=f"{min_len}-{max_len} chars"):
        samples.append(generate_text(min_len, max_len))

# ---- Save ----
output_file = "gemini_immigration_texts.csv"

with open(output_file, mode="w", newline="", encoding="utf-8") as f:
    writer = csv.DictWriter(
        f,
        fieldnames=["model", "target_length", "actual_length", "text"]
    )
    writer.writeheader()
    writer.writerows(samples)

print(f"Saved {len(samples)} texts to {output_file}")
```