

TopicBench

Benchmarking LLMs for Topic Labeling

BE BOUNDLESS

W

Background

- > **The Problem:** Researchers use topic modeling to find themes in text
- > **The Challenge:** These themes need human-readable labels. Which AI model is best?
- > **Traditional Approach:** Manually label topics (slow, subjective)
- > **Our Approach with TopicBench:** Automatically compare different LLMs and pick the best one

Data Used

- > 5 research papers across 5 different fields
 - Sociology
 - Medicine
 - Environmental Science
 - Human-computer interaction (HCI)
 - Natural language processing (NLP)
- > **Data type:** Keywords from topic modeling + human interpretations

Example of Data

Paper Title	Field	Keywords	Author's Label	Human Labeling
almquist_bagozzi_2019	sociology	[["one", "made", "anoth'..."]]	['Inspirational Language', 'Group Identity Debates', 'Neocolonialism'...]	['Motivational Rhetoric', 'Collective Identity Discussions', 'Neo-imperialism'...]
dinsa_2024	medicine	[["body", "came", "dries up'..."]]	['Nervous disease', 'Gynecology', 'Mental illness'...]	['Fatigue & General Body Weakness', 'Pregnancy & Menstrual Health', 'Head, Ear & Respiratory Issues'...]

etc etc...

Use Cases

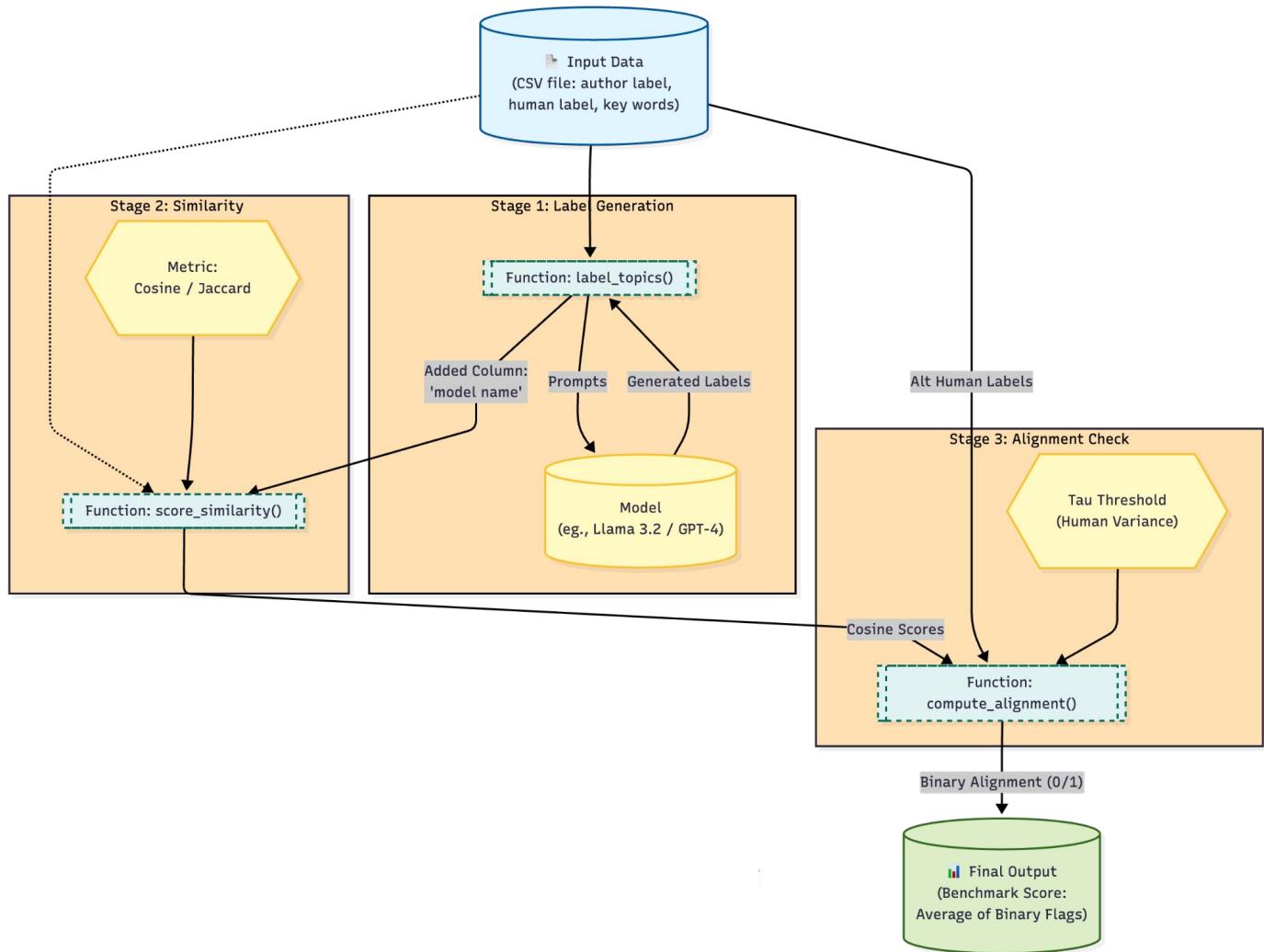


- **User:** John, a computational social scientist analyzing large text datasets.
- **Goal:**

Identify topic patterns and select the LLM that produces the most accurate topic labels.
- **Constraints:**
 - Works with sensitive data; cannot use cloud APIs.
 - Needs reproducible, local evaluation.
 - Cannot write API integration code for multiple models.
- **How TopicBench Helps:**
 1. Label Generation:
 - Runs topic labeling locally using different LLMs.
 - Records model name for comparison.
 2. Similarity Scoring:
 - Computes cosine / Jaccard similarity for author, human, and LLM labels.
 3. Alignment Check:
 - Applies threshold to classify model alignment (0/1).
- **Outcome:**

TopicBench allows John to choose the most accurate LLM for research while keeping data secure.

Design



Demo

Import:

```
import pandas as pd
import numpy as np
import os
import ast
import json
from tqdm import tqdm
from src.topicbench.label_topics import label_topics
from src.topicbench.validate import score_similarity, compute_alignment
```



Label generation + similarity score

field	keywords	author_label	alt_human	llama3.2:latest	llama3.2:latest_cosine_similarity	alt_human_cosine_similarity
sociology	[['one', 'made', 'anoth', 'everi', 'side', 'ti...]	['Inspirational Language', 'Group Identity Deb...']	['Motivational Rhetoric', 'Collective Identity...', 'Protest', 'Re...']	['Social Movement', 'Activism', 'Protest', 'Re...']	[0.22654280066490173, 0.4005390405654907, 0.34...]	[0.4601529836654663, 0.857661247253418, 0.7370...]
medicine	[['body', 'came', 'dries up', 'rocking', 'time...']	['Nervous disease', 'Gynecology', 'Mental illn...']	['Fatigue & General Body Weakness', 'Pregnancy...', 'Menstrual Cycle', ...]	['Gastrointestinal Issues', 'Menstrual Cycle', ...]	[0.15294265747070312, 0.41162168979644775, 0.3...]	[0.2469634711742401, 0.31719857454299927, 0.11...]

Benchmark score for different model:

	field	author_label	alt_human	llama3.2:latest	gpt-3.5-turbo	llama3.2:latest_final_score	gpt-3.5-turbo_final_score
0	sociology	['Inspirational Language', 'Group Identity Deb...']	['Motivational Rhetoric', 'Collective Identity...', 'Protest', 'Resistance', ...]	['Social Movement', 'Activism', 'Protest', 'Resistance', ...]	['movement and activism', 'environmental conse...']	0.600000	0.333333
1	medicine	['Nervous disease', 'Gynecology', 'Mental illn...']	['Fatigue & General Body Weakness', 'Pregnancy...', 'Menstrual Cycle', ...]	['Gastrointestinal issues', 'Pregnancy complic...']	['Physical Symptoms', 'Pregnancy-related Issue...']	0.222222	0.200000

```
import pandas as pd
import numpy as np
import os
import ast
import json
from tqdm import tqdm
from src.topicbench.label_topics import label_topics
from src.topicbench.validate import score_similarity, compute_alignment

# specify the models that will be benchmarked
models_to_benchmark = [
    (llama3.2.latest, {'model_key_path': None, 'type': 'local'}),
    (gpt-3.5-turbo, {'model_key_path': None, 'type': 'local'}),
    (llama3.2.latest, {'model_key_path': 'https://api.topicbench.com/api_key/part1'}, 'remote'),
    (gpt-3.5-turbo, {'model_key_path': 'https://api.topicbench.com/api_key/part1'}, 'remote')
]

# and the dataset to use
dataset = 'TopicBench'

# define parse_label(label_string)
def parse_label(label_string):
    """ Safely parse label strings that may contain apostrophes or other special characters. """
    try:
        parsed_label = ast.literal_eval(label_string)
    except ValueError, SyntaxError:
        try:
            # In case fails, try replacing single quotes with double quotes
            parsed_label = eval(label_string.replace("'", '"'))
        except SyntaxError:
            print("Warning: Could not parse: " + label_string)
    except json.decoder.JSONDecodeError:
        print("Warning: Could not parse: " + label_string)
    return parsed_label

# for each model, we will compute similarity scores
for model_name in models_to_benchmark:
    all_scores = []
    for row in dataset:
        row = results_df.loc[row]
        # Parse author_label and alt_label
        author_label = parse_label(row['author_label'])
        alt_label = parse_label(row['alt_label'])

        # Compute cosine similarity between corresponding labels
        if len(author_label) == len(alt_label):
            label_similarities = []
            for i in range(len(author_label)):
                label_similarities.append(score_similarity(author_label[i], alt_label[i], metric='cosine'))
            total_similarity.append(label_similarities)
            all_scores.append(label_similarities)

    # Add the mean of the column with format: model_name_mean_scores
    results_df[f'{model_name}_mean_scores'] = all_scores

# calculate mean similarity for all models
results_df['all_mean_scores'] = results_df.mean(axis=1)

# calculate mean similarity for each model
for model_name in models_to_benchmark:
    all_scores = []
    for row in dataset:
        row = results_df.loc[row]
        # Parse author_label and alt_label
        author_label = parse_label(row['author_label'])
        alt_label = parse_label(row['alt_label'])

        # Compute cosine similarity between corresponding labels
        if len(author_label) == len(alt_label):
            label_similarities = []
            for i in range(len(author_label)):
                label_similarities.append(score_similarity(author_label[i], alt_label[i], metric='cosine'))
            total_similarity.append(label_similarities)
            all_scores.append(label_similarities)

    # Add the mean of the column with format: model_name_mean_scores
    results_df[f'{model_name}_mean_scores'] = all_scores

print("Similarity computations complete!")

results_df.to_csv('TopicBench.csv')

# calculate alignment scores for each model in the list for all models
for model_name in models_to_config.keys():
    alignment_scores = []
    for row in dataset:
        row = results_df.loc[row]
        # Parse author_label and alt_label
        author_label = parse_label(row['author_label'])
        alt_label = parse_label(row['alt_label'])

        # Compute alignment scores
        if len(author_label) == len(alt_label):
            alignment_scores.append(score_alignment(author_label, alt_label))
        else:
            alignment_scores.append(0.0)

    # Add the mean of the column with the individual scores
    results_df[f'{model_name}_mean_scores'] = alignment_scores
    results_df[f'{model_name}_mean_alignment_scores'] = alignment_scores

print("Alignment computations complete!")

# calculate alignment scores for each model in the list for all models
for model_name in models_to_config.keys():
    alignment_scores = []
    for row in dataset:
        row = results_df.loc[row]
        # Parse author_label and alt_label
        author_label = parse_label(row['author_label'])
        alt_label = parse_label(row['alt_label'])

        # Compute alignment scores
        if len(author_label) == len(alt_label):
            alignment_scores.append(score_alignment(author_label, alt_label))
        else:
            alignment_scores.append(0.0)

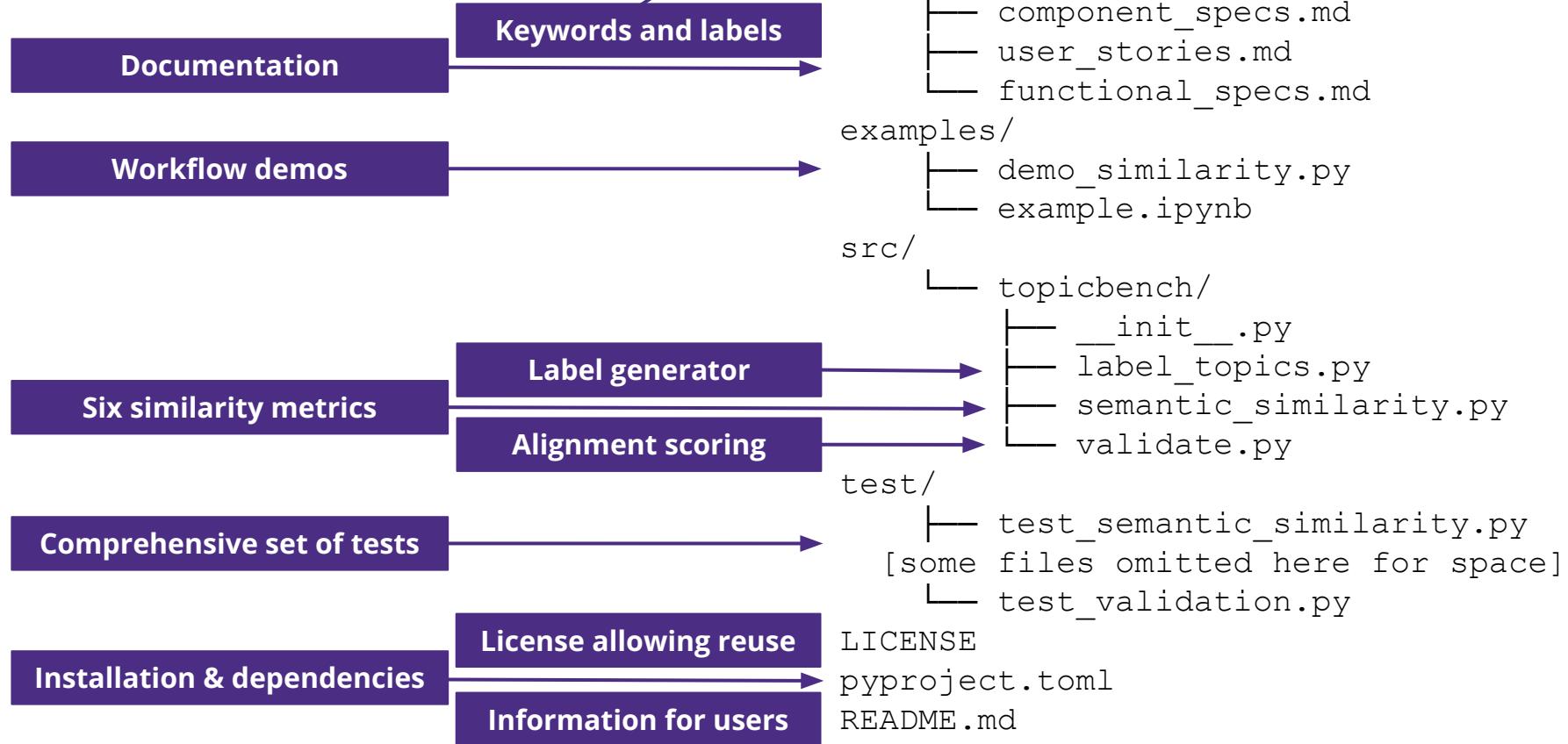
    # Add the mean of the column with the individual scores
    results_df[f'{model_name}_mean_scores'] = alignment_scores
    results_df[f'{model_name}_mean_alignment_scores'] = alignment_scores

print("Alignment computations complete!")

# calculate the mean of alignment scores for each model in the list for all models
for model_name in models_to_config.keys():
    alignment_mean_scores = results_df[f'{model_name}_mean_alignment_scores'].mean()
    print(f'Alignment mean for {model_name}: {alignment_mean_scores}')

# calculate the mean of alignment scores for each model in the list for all models
for model_name in models_to_config.keys():
    alignment_mean_scores = results_df[f'{model_name}_mean_alignment_scores'].mean()
    print(f'Alignment mean for {model_name}: {alignment_mean_scores}')
```

Project Structure



Lessons Learned & Future Work

> Lessons Learned

- Clear separation between labeling, similarity scoring, and alignment improves design.
- Modular code and documentation made it easier for the whole team to collaborate

> Future Work

- **User Interface:** Add a simple web or notebook-based UI so non-technical researchers can run benchmarks easily.
- **Custom Datasets:** Allow users to upload their own datasets + automated preprocessing pipeline
- **Model Integrations:** Support additional LLM providers and custom local models.