

✓ Case Study Part 2: Insights

This notebook is running on a Google Colab L4 CPU environment. In this section, the metrics inferred from the data in part 1 of the case study are clustered, analyzed and visualized.

```
1 %%capture
2 !pip install numpy --upgrade #(uncomment install lines if running first time in google colab)
3 !pip install scipy --upgrade
4 !pip install gensim #(refresh kernel after installing to load new library and recoment installs)

1 %%capture
2 import pandas as pd
3 import gensim #gensim is used for a latent text embedding model for clustering text by meaning
4 from gensim.models import LdaMulticore
5 from gensim.corpora import Dictionary
6 from gensim.models.coherencemodel import CoherenceModel
7 from matplotlib import pyplot as plt
8 import matplotlib.patches as mpatches
9 !pip install wordcloud
10 from wordcloud import WordCloud

1 from gensim.models.phrases import Phrases
2 from sklearn.feature_extraction.text import TfidfVectorizer
3 #from sklearn.cluster import KMeans
4 #from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score, silhouette_score
5 import numpy as np

1 %%capture
2 !pip install datasets
3 from datasets import load_dataset
4 from google.colab import userdata

1 data = load_dataset("skeskinen/TinyStories-GPT4")

1 data_df = data['train'].to_pandas()

1 word_occurrences = pd.read_csv('word_occurrences.csv')

1 feature_occurrences = pd.read_csv('feature_occurrences.csv')
```

✓ Statistical Analysis and Visualization

In this section the statistical metrics calculated from the training data is analyzed and visualized for making insights with regards to the stated objectives. Furthermore, given the complexity in understanding the level of literacy from the data - as no 'literacy level' column is within the data - a language model is used to classify the responses.

✓ Mental Health

As for mental-health, goals may be defined in objective terms by inferring the emotions, subjects, topics, genres, or any commonly used keywords related to mental-health within the data. For this purpose, we can use the list of words and narrative features provided within the prompts provided to the AI in the training data for each row in the dataset. By counting the occurrences of the 'words' and 'features' columns used within the prompt, we can make some insights of the mental-health objectives.

Objective: Highlight narrative features that relate to mental health.

✓ Narrative Features

Some of the 'feature narrative' words provided to the model within the prompt template's in the training dataset give insight into whether the story may contain any emotional subjects or topics related to mental health. A bar plot is shown that highlights the amount of each feature

narrative.

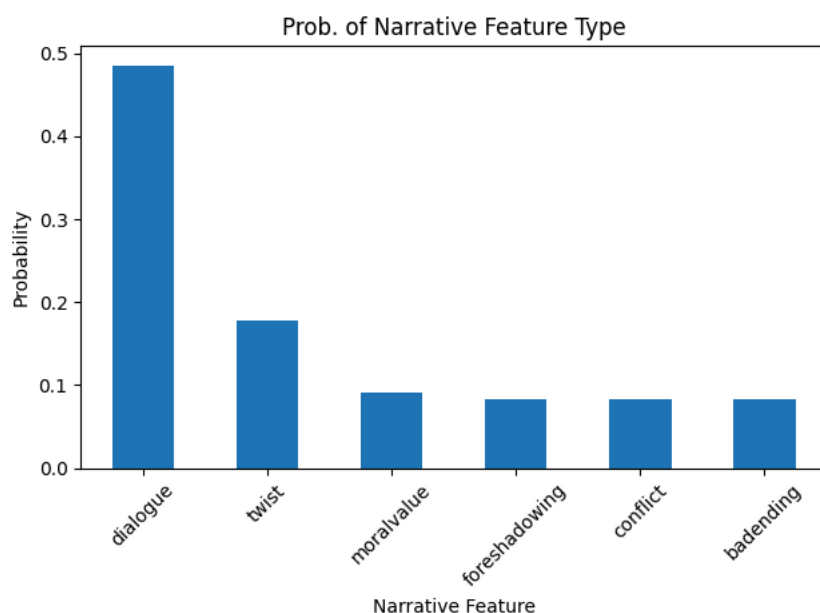
```
1 # converting the counts of feature occurrences to probability values
2 feature_occurrence_probabilities = feature_occurrences['feature_occurrences'] / feature_occurrences['feature_occurrences'].sum()

1 feature_occurrence_probabilities.index = feature_occurrences['feature']
```


✓ Mental Health Insight

We can see from the bar chart that the majority of narratives have dialogue (i.e speech between characters in a story). Similarly, 'twists' and 'moralvalue' are 2nd and 3rd respectively, highlighting how the majority of prompts seek to generate interesting stories where characters interact and exemplify moral values. On the other end of the spectrum, the story prompts in the training data generate stories with narratives like foreshadowing, conflict, and bad endings. As our target demographic is youth, it might be a good idea to do further investigation on story prompts that contain the 'conflict' and 'badending' narrative tags - these combined tags are contained in roughly **20%** of the prompts.

```
1 ax = feature_occurrence_probabilities.sort_values(ascending=False).plot(
2     kind='bar',
3     title='Prob. of Narrative Feature Type'
4 )
5
6 # Correctly set labels using set methods
7 plt.ylabel('Probability')
8 plt.xlabel('Narrative Feature')
9 plt.xticks(rotation=45)
10
11 # Optional: Adjust layout to prevent label cutoff
12 plt.tight_layout()
13
14 plt.show()
```



```
1 feature_occurrences.describe()
```


feature_occurrences

count	6.000000e+00
mean	5.059540e+05
std	4.859297e+05
min	2.503000e+05
25%	2.507192e+05
50%	2.624705e+05
75%	4.730752e+05
max	1.470404e+06



1 feature_occurrence_probabilities

**feature_occurrences****feature**

dialogue	0.484367
moralvalue	0.090309
twist	0.177679
foreshadowing	0.082613
badending	0.082452
conflict	0.082582

dtype: float64

```

1 # Sum of 'badending' & 'conflict'
2 badending_conflict_narrative_probability = feature_occurrence_probabilities.loc[
3     feature_occurrence_probabilities.index.isin(['badending', 'conflict'])
4 ].sum()
5
6 # Sum of dialogue, twist, and foreshadowing
7 dialogue_twist_foreshadowing_narrative_probability = feature_occurrence_probabilities.loc[
8     feature_occurrence_probabilities.index.isin(['dialogue', 'twist', 'foreshadowing'])
9 ].sum()
10
11 # Summing mental health related narratives
12 mental_health_narrative_probability = badending_conflict_narrative_probability + feature_occurrence_probabilities.loc['moralv

```

```

1 def plot_topic_counts(df, title="Topics", xlabel="Topic", ylabel="Probability", pre_normalized=False):
2     if not pre_normalized:
3         # Getting topic counts and labels
4         topic_counts = df['topic'].value_counts(normalize=True)
5     else:
6         topic_counts = df['probability'].sort_values(ascending=False)
7     labels = topic_counts.index.tolist()
8     counts = topic_counts.values.tolist()
9
10    # Plotting the bars with distinct colors
11    # Generating distinct colors using a colormap
12    numBars = len(labels)
13    cmap = plt.get_cmap('viridis', numBars) # Choose a colormap
14
15    bars = plt.bar(labels, counts, color=[cmap(i) for i in range(numBars)])
16
17    # Setting chart elements
18    plt.title(title)
19    # Hiding x-ticks and labels
20    plt.xticks([])
21    plt.ylabel(ylabel)
22    #plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better visibility
23    plt.tight_layout() # Prevents labels from overlapping
24
25    # Generating legend handles with x-axis labels as legend entries
26    legend_handles = [mpatches.Patch(color=cmap(i), label=label) for i, label in enumerate(labels)]
27
28    # Smaller legend with adjusted position

```

```

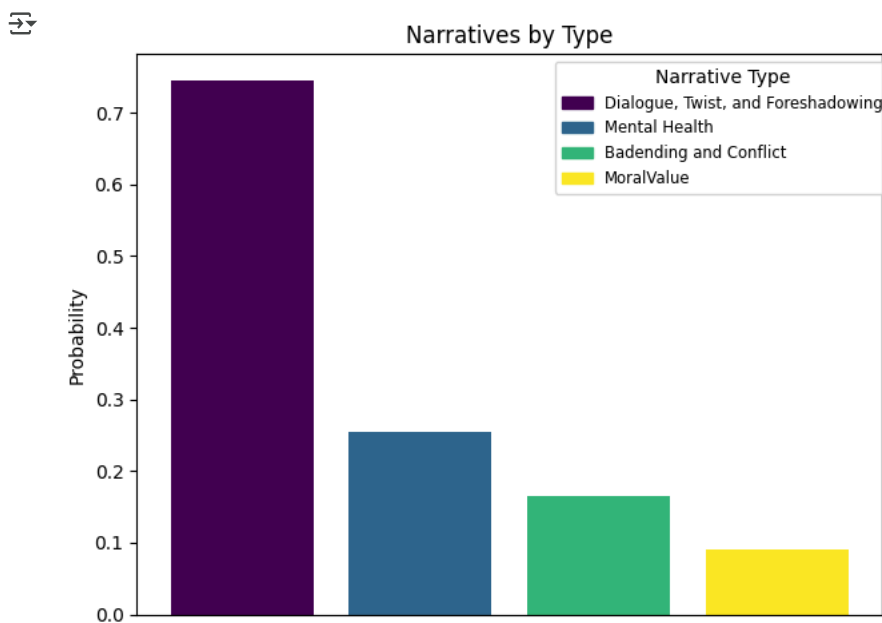
29 plt.legend(handles=legend_handles, title=xlabel, loc='upper right', fontsize='small', bbox_to_anchor=(1.02, 1))
30
31 plt.show()

1 print(f"Probability of Mental Health Narrative: {mental_health_narrative_probability}")
2 print(f"Probability of Dialogue, Twist, and Foreshadowing Narrative: {dialogue_twist_foreshadowing_narrative_probability}")
3 print(f"Probability of Badending and Conflict Narrative: {badending_conflict_narrative_probability}")
4 print(f"Probability of MoralValue Narrative: {mental_health_narrative_probability - badending_conflict_narrative_probability}")
5 # making a df of these probabilities
6 narrative_probabilities_df = pd.DataFrame({
7     'topic': ['Mental Health', 'Dialogue, Twist, and Foreshadowing', 'Badending and Conflict', 'MoralValue'],
8     'probability': [mental_health_narrative_probability, dialogue_twist_foreshadowing_narrative_probability, badending_conflict_narrative_probability, mental_health_narrative_probability - badending_conflict_narrative_probability]
9 })
10 # make into series by making topic the index
11 narrative_probabilities_df = narrative_probabilities_df.set_index('topic')

↗ Probability of Mental Health Narrative: 0.2553420534936641
Probability of Dialogue, Twist, and Foreshadowing Narrative: 0.7446579465063359
Probability of Badending and Conflict Narrative: 0.1650334483635346
Probability of MoralValue Narrative: 0.09030860513011066

```

```
1 plot_topic_counts(narrative_probabilities_df, title='Narratives by Type', xlabel='Narrative Type', ylabel='Probability', pre
```



Statistical Insight

The description of the feature occurrences show that the average count of prompts with each narrative type is typically ~500,000 for each. However, the 50% quantile is ~260,000 showing that the majority of narrative types are roughly half as large as the mean. This implies that there are outlier narratives skewing the distribution up. The 'dialogue' and 'twist' narrative types are outliers, and are contained within ~50% and ~20% of the story prompts respectively, and combined contribute to ~70% of the story prompts.

The plot of the narrative features by type show that the non-mental-health related narratives dominate the narrative features in the sampled prompts with ~75% of the narratives. Mental health narratives amount to ~25% of the rest of the narratives. **Within the mental health narrative category, the combined 'badending and conflict' narratives outweigh the 'moralvalue' narratives by almost double.**

Note: Most of the prompts range in number of narrative tags, and may contain either none or all of the tags described above. As such, it is worth investigating how these tags correlate. For example, a combined 'badending' and 'conflict' narrative lacking a 'moralvalue' tag might lead to risky or insensitive storytelling.

These insight are made with only a small sample of the overall training data.

Key Words

The key words column 'words' in the data are specific key vocabulary words provided to the AI in the story generation prompt. These words differ from the narrative feature words as the key words must be included in the generated story - whereas the narrative features are abstract storytelling features.

```

1 # converting the counts of feature occurrences to probability values
2 word_occurrence_probabilities = word_occurrences['word_occurrences'] / word_occurrences['word_occurrences'].sum()

1 word_occurrence_probabilities.index = word_occurrences['word']

1 word_occurrence_probabilities.describe()

```

↗

word_occurrences	
count	1603.000000
mean	0.000624
std	0.000422
min	0.000294
25%	0.000311
50%	0.000319
75%	0.000854
max	0.002497

dtype: float64

As the collection of vocabulary words within the dataset is large (~1604), the data must be clustered along the words' sentiment/meaning using a text embedding model. This way, we can reduce the dimensionality of the data while somewhat preserving the intended meaning of the collection. Given the individual probabilities for each word are very small (on the scale of 1/1000 likeliness), clustering to reduce the dimensionality will help understand the salient groups of words that are commonly paired within each vocab key collection.

```

1 # step 1: prepare the n-grams
2 data_df['words'].head() # every collection of words is already of length 3 (trigram format)

```

↗

words	
0	[receive, opera, red]
1	[use, sheet, blue]
2	[relax, bus, uncomfortable]
3	[sail, cricket, wide]
4	[pray, pigeon, creative]

dtype: object

Since the dataset is large, clustering can not be executed in one execution - even with Google Colabs biggest ram environment. Therefore, the data will be clustered in iterations via bootstrapping. In this way, the data will be clustered in batches and then aggregated.

```

1 # note: vectorizer must be initialized outside the loop
2 # in order to maintain the training data across batches
3
4 # Step 2: LDA Clustering the batched TF-IDF matrix with
5 def bootstrap_clustering(data, num_clusters, num_iterations, batch_size):
6     """
7     Performs bootstrapped clustering on the data, incorporating TF-IDF creation within the loop.
8
9     Args:
10         data: The data to cluster (the 'words' column of your DataFrame).
11         num_topics: The number of clusters in the LDA clustering.
12         num_iterations: The number of iterations.
13         batch_size: The batch size.
14
15     Returns:
16         A list of cluster assignments for each data point.
17     """
18
19     all_topic_assignments = []
20     batches = []
21     topic_assignment_batches = []
22     # Initialize lists to store cluster statistical metrics

```

```

23 coherence_scores = []
24 perplexity_scores = []
25
26 data_size = len(data)
27
28 for _ in range(num_iterations):
29     # Randomly sample a batch of data
30     batch_indices = np.random.choice(data_size, size=batch_size, replace=True)
31     batch_data = data.iloc[batch_indices]['words'] # Select the 'words' column
32     batches.append(batch_data)
33
34     # Prepare data for LDA
35     dictionary = Dictionary([word for word in batch_data.values])
36     corpus = [dictionary.doc2bow(text) for text in batch_data.values]
37
38     # Perform LDA on the batch
39     lda_model = LdaMulticore(corpus, num_topics=num_topics, id2word=dictionary, passes=15, workers=2)
40     batch_topic_assignments = [max(lda_model[doc], key=lambda item: item[1])[0] for doc in corpus]
41
42     topic_assignment_batches.append(batch_topic_assignments)
43     # Append the cluster assignments to the list
44     all_topic_assignments.extend(batch_topic_assignments)
45     # Calculate and store metrics for this iteration
46     coherence_model = CoherenceModel(model=lda_model, texts=batch_data.values, dictionary=dictionary, coherence='c_v')
47     coherence_score = coherence_model.get_coherence()
48     coherence_scores.append(coherence_score)
49
50     perplexity = lda_model.log_perplexity(corpus)
51     perplexity_scores.append(perplexity)
52
53 return all_topic_assignments, batches, topic_assignment_batches, coherence_scores, perplexity_scores

1 num_topics = 12 # 12 is chosen to be able to visualize wordcloud in grid format
2 num_iterations = 10 # Adjust as needed
3 batch_size = 1500 # This number is close to the maximum compute
4
5 # Perform bootstrapped clustering
6 topic_assignments, batches, topic_assignment_batches, coherence_scores, perplexity_scores = bootstrap_clustering(
7     data_df, num_topics, num_iterations, batch_size
8 )
9
10 # Assign the cluster assignments to the original DataFrame
11 # word_trigram_df['cluster'] = cluster_assignments

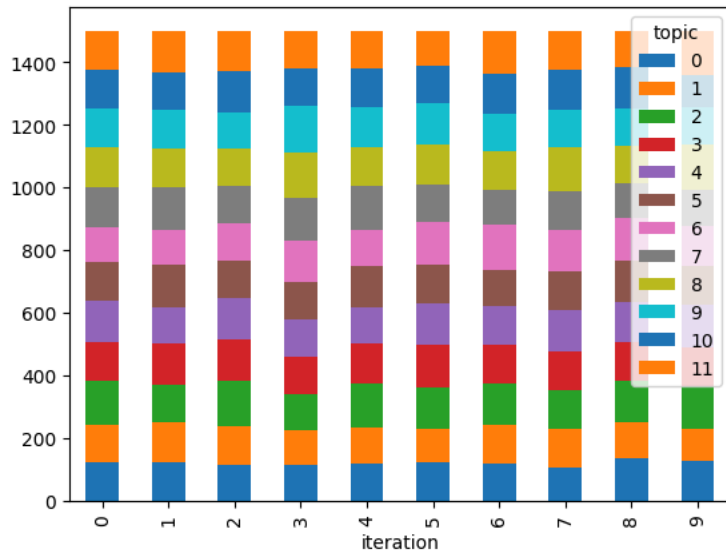
1 cluster_assignments_df = pd.DataFrame(topic_assignments, columns=['topic'])
2 # adding in a cluster iteration index
3 cluster_assignments_df['iteration'] = cluster_assignments_df.index // batch_size

1 cluster_counts_by_iteration = cluster_assignments_df.groupby('iteration')['topic'].value_counts()

1 # let's visualize the distribution of cluster counts by iteration by showing a bar chart with a legend using the iteration
2 cluster_counts_by_iteration.unstack().plot(kind='bar', stacked=True, legend=True)

```

<Axes: xlabel='iteration'>



Since the topics discovered via LDA clustering are abstract, classification of these topics into a understandable theme may be done using a large language model with a prompt to classify into the existing objectives of 'mental health' or 'creativity'.

```

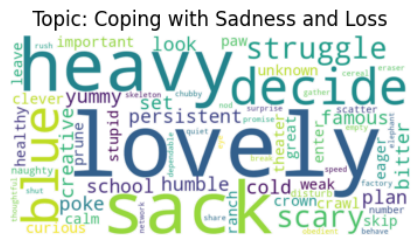
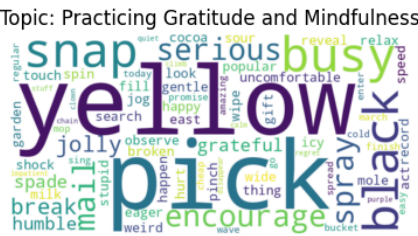
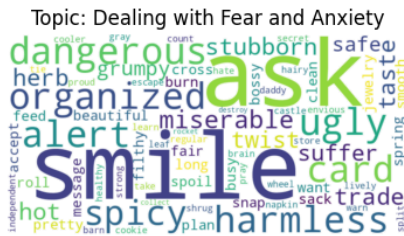
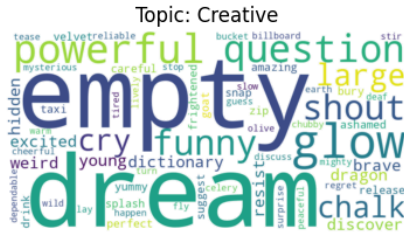
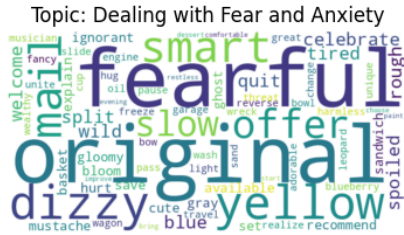
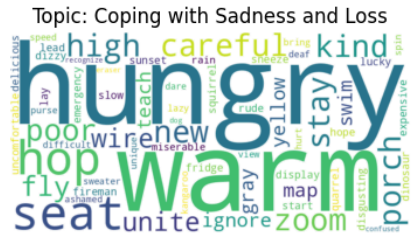
1 mental_health_topics = [
2     "Dealing with Fear and Anxiety",
3     "Coping with Sadness and Loss",
4     "Managing Anger and Frustration",
5     "Practicing Gratitude and Mindfulness",
6     "Developing Healthy Relationships with Family",
7     "Developing Healthy Relationships with Friends"
8 ]
9
10 creativity_topics = [
11     "Thinking Outside the Box",
12     "Embracing Imagination and Curiosity",
13     "Using Art and Expression",
14     "Music",
15     "Painting",
16     "Sculpting"]

```

```

1 # let's create a function to call openAI's chatgpt-4o with a prompt to classify topics
2 import openai
3 import os
4
5 client = openai.OpenAI(api_key=userdata.get('openai'))
6 model = 'gpt-4o-mini'
7 def classify_topics(mental_health_topics, creativity_topics, topic_df_words_counts):
8     """ this takes a preset list of mental health and creativity topics,
9         and classifies the topic_ids by using the data in the topic_df_words
10        note: there are 12 topic_ids, and 6 mental health topics, and 6 creativity topics
11        thus only one topic_id can be classified into one of the 2 categories
12    """
13    prompt_template = (
14        f"The following is a list of mental health topics: {mental_health_topics}",
15        f"The following is a list of creative topics: {creativity_topics}",
16        f"Use this list of words and their counts to classify the most likely topic: {topic_df_words_counts}",
17        "Note: Each topic in either list can be assigned only once. Only respond with the topic chosen and nothing else. The
18        Topic: "
19    )
20    prompt = "\n".join(prompt_template)
21    response = client.chat.completions.create(
22        messages=[
23            {"role": "system", "content": "You are comfortable classifying the language used in stories by simply inferring
24            {"role": "user", "content": prompt}
25        ],
26        model = model,
27        temperature = 0.0
28    )
29    return response.choices[0].message.content
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

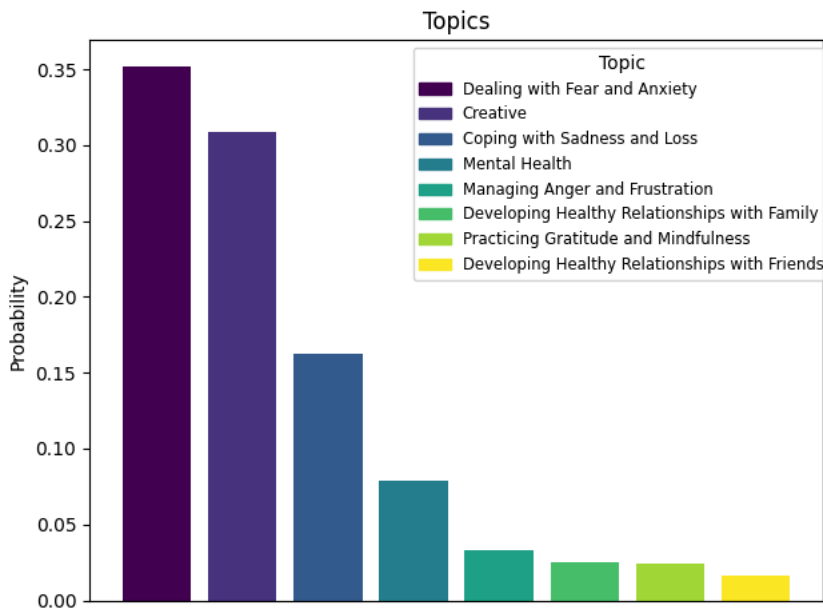
```

1 # Step 4: classifying all the topic_ids across all bootstrap cluster iterations
2 all_classified_topic_batches = []
3 for i in range(len(topic_assignment_batches)):
4     batch_df = pd.DataFrame({'words': batches[i], 'topic': topic_assignment_batches[i]})
5     for topic_id in batch_df['topic'].unique():
6         topic_df = batch_df[batch_df['topic'] == topic_id].copy()
7         topic_df['words'] = topic_df['words'].apply(lambda x: ' '.join(x))
8         # Concatenate all word strings in topic_df['words']
9         wordcloud_text = ' '.join(topic_df['words'].values)
10        # Filter counts of all words in the wordcloud_text and then select top 100 with count > 1
11        wordcloud_text_counts = wordcloud.process_text(wordcloud_text)
12        wordcloud_text_counts = {k: v for k, v in sorted(wordcloud_text_counts.items(), key=lambda item: item[1], reverse=True)}
13        wordcloud_text_counts = dict(list(wordcloud_text_counts.items())[:100])
14        wordcloud_text_counts = pd.Series(wordcloud_text_counts, index=wordcloud_text_counts.keys())
15        # Filter to words with counts > 1
16        wordcloud_text_counts = wordcloud_text_counts[wordcloud_text_counts > 1]
17        topic = classify_topics(mental_health_topics, creativity_topics, wordcloud_text_counts.to_string())
18        topic_df['topic'] = topic
19        all_classified_topic_batches.append(topic_df)

1 # concatenating all tables for summary
2 all_classified_topic_batches_df = pd.concat(all_classified_topic_batches)

1 plot_topic_counts(all_classified_topic_batches_df)

```



✓ Mental Health Insight

The classification of the topic id's found via clustering of the key words using a Latent Dirichlet Allocation model for latent text embedding analysis is shown above. It is surprising that a majority of the classified topic labels fall under the 'mental health' category. However, despite the language model not classifying much of the sub-categories within the 'creativity' category, it still categorized much of the clusters as belonging to the general 'creative' label.

Note: This insight is made with only a small sample of the overall training data.

```

1 # summing the probability of all labels except 'Creative'
2 mental_health_probability = 1 - all_classified_topic_batches_df['topic'].value_counts(normalize=True)['Creative']
3 creative_probability = 1 - mental_health_probability
4 print(f"Probability of Mental Health: {mental_health_probability}")
5 print(f"Probability of Creative: {creative_probability}")

```



```

Probability of Mental Health: 0.6915333333333333
Probability of Creative: 0.3084666666666667

```

Statistical Insight

A majority of the vocabulary data clusters sampled (~70%) were assigned to a variety of 'mental health' topic categories. Compared to the 'creative' topic, which only amounts to ~30% of the sampled clusters and has almost no variety in labels, it seems as if the training data provides more mental-health related key vocabulary words. This might be correlated to the influence of the 'narrative' feature category of 'badending' and 'conflict', with smaller influence due to 'moralvalue'. **It is important to note that despite this, all of the mental health narratives amount to only ~25% of the full training data and that the 'moralvalue' narrative is almost half of that of the combined 'badending' and 'conflict' narratives.**

Note: This insight is made with only a small sample of the overall training data.

✓ Creativity

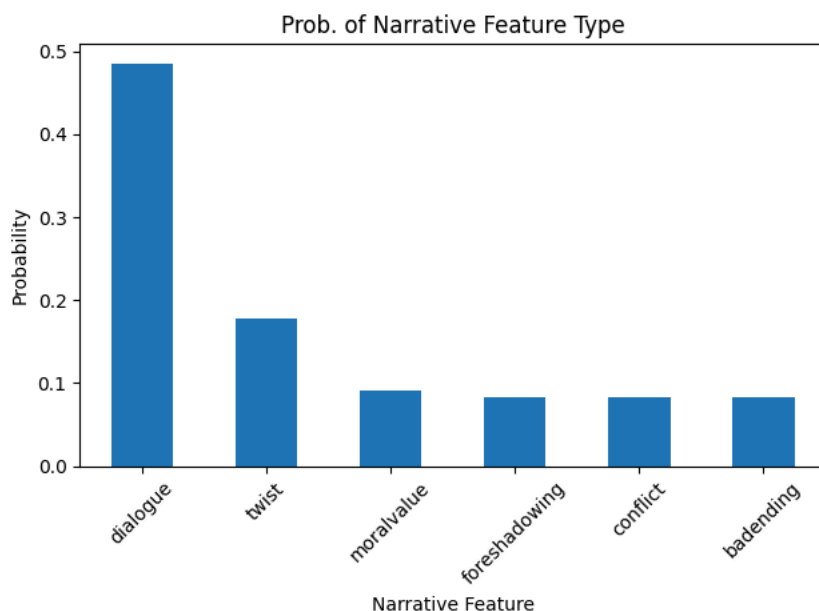
Objectively speaking, creativity is a hard goal to define as it can be objectively defined in a multitude of way, as are the aforementioned topics of literacy and mental-health. However, many people might agree that creativity is somehow unique. Therefore, it may be possible to define the goal of creativity by understanding the level of variance in the models responses to similar prompts.

✓ Narrative Features

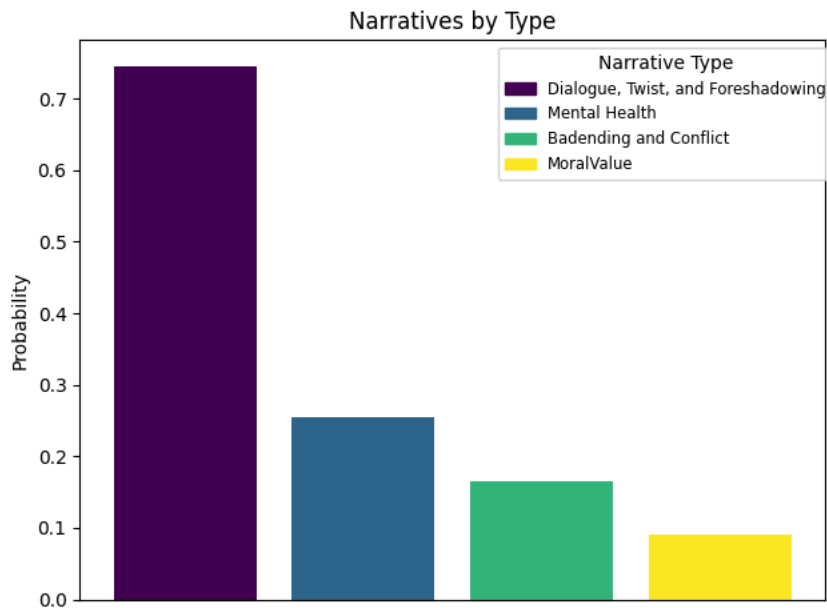
✓ Creativity Insight

The bar plot below shows how the narratives that were found fall under narrative types belonging to literacy (dialogue, twist, foreshadowing) or mental health (bad ending, conflict, moral value). Despite containing 6 narrative types, a majority of the narrative types belong to only a few narratives (with dialogue and twist amounting to ~70% of all narratives combined). Similarly, 'dialogue', 'twist', and 'foreshadowing' are narratives that are less creative as they do not necessarily influence the plot of the story itself and moreso influence the storytelling structure.

```
1 ax = feature_occurrence_probabilities.sort_values(ascending=False).plot(
2     kind='bar',
3     title='Prob. of Narrative Feature Type'
4 )
5
6 # Correctly set labels using set methods
7 plt.ylabel('Probability')
8 plt.xlabel('Narrative Feature')
9 plt.xticks(rotation=45)
10
11 # Optional: Adjust layout to prevent label cutoff
12 plt.tight_layout()
13
14 plt.show()
```



```
1 plot_topic_counts(narrative_probabilities_df, title='Narratives by Type', xlabel='Narrative Type', ylabel='Probability', pre
```



Statistical Insight

Thus, there is a need to increase the variety of narratives, as well as to improve the distribution by removing the bias for the dialogue outlier narrative. By looking at the specific probabilities of the more creative narrative types, such as 'moralvalue', 'badending', and 'conflict' (all falling under the mental health narrative umbrella), we can gain insight into the kind's of plots featured within the stories. **The subjectively more creative mental health narrative umbrella type corresponds to ~25% of the narratives in the full training data. Thus, there is a need to increase the amount of creative narratives within the training data.**

```
1 print(f"Probability of Mental Health Narrative: {mental_health_narrative_probability}")
2 print(f"Probability of Dialogue, Twist, and Foreshadowing Narrative: {dialogue_twist_foreshadowing_narrative_probability}")
3 print(f"Probability of Badending and Conflict Narrative: {badending_conflict_narrative_probability}")
4 print(f"Probability of MoralValue Narrative: {mental_health_narrative_probability - badending_conflict_narrative_probability}")
```



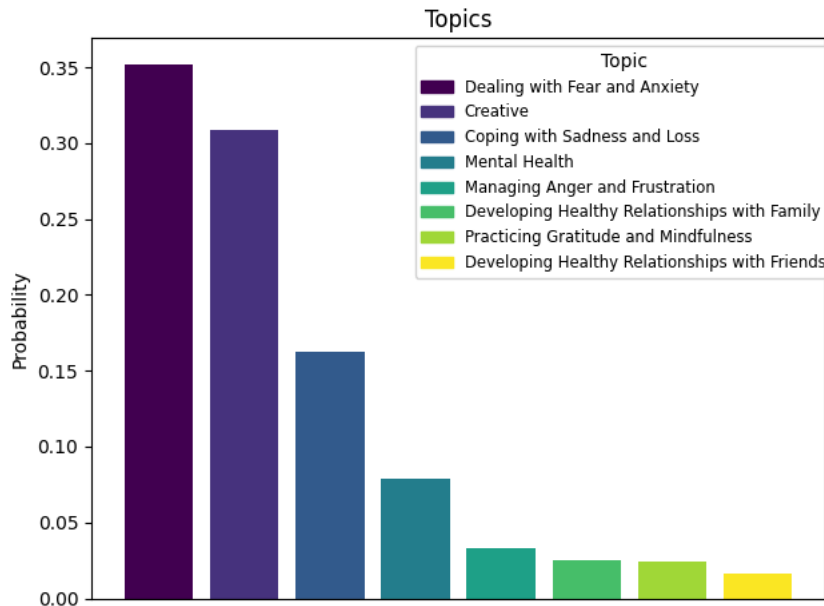
```
Probability of Mental Health Narrative: 0.2553420534936641
Probability of Dialogue, Twist, and Foreshadowing Narrative: 0.7446579465063359
Probability of Badending and Conflict Narrative: 0.16503344836355346
Probability of MoralValue Narrative: 0.09030860513011066
```

Key Words

Creativity Insight

In the barplot, we can see that the creativity and mental health topics classified using the language model are mostly highlighting the 'mental health' category and lack the subcategories that were provided for the 'creativity' category. Despite the lack of sufficient description of the kinds of 'creativity', 'creativity' as a general label for the latent text embedding clusters is still the second-most popular label.

```
1 plot_topic_counts(all_classified_topic_batches_df)
```



Statistical Insight

The cumulative probability for the mental health topic category for the vocabulary key words far outweighs that of the singular 'creative' label, despite the 'creative' label being the second-most popular. As shown in the above probability values, 'creativity' is only ~30% of the vocabulary word clusters. Future prompt generation should potentially focus on including more prompts that highlight creative activities in the stories such as the 'painting', 'music', or 'sculpting' categories provided to the language model for classification that were ultimately not assigned during classification.

```
1 # summing the probability of all labels except 'Creative'
2 mental_health_probability = 1 - all_classified_topic_batches_df['topic'].value_counts(normalize=True)['Creative']
3 creative_probability = 1 - mental_health_probability
4 print(f"Probability of Mental Health: {mental_health_probability}")
5 print(f"Probability of Creative: {creative_probability}")
```

Probability of Mental Health: 0.6915333333333333
Probability of Creative: 0.3084666666666667

Literacy

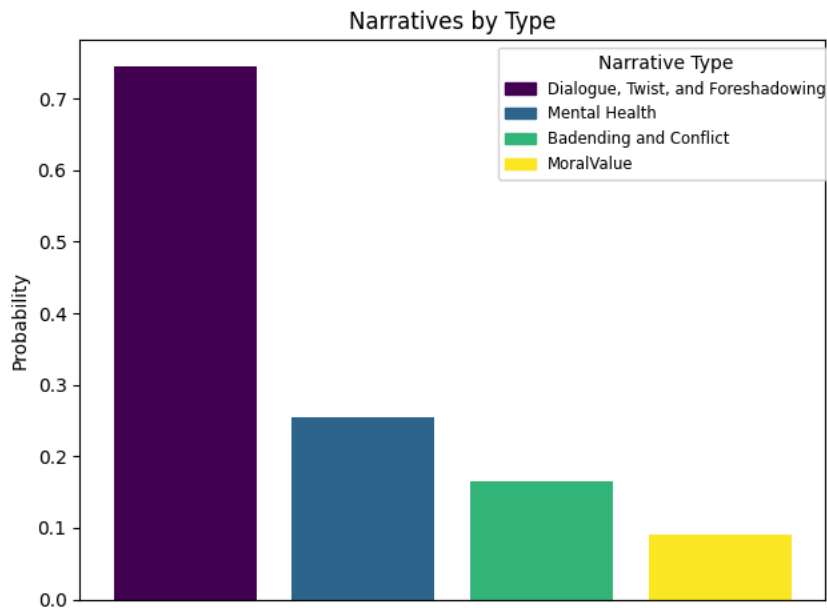
WonderWords' literacy goals may be defined in objective terms by classifying whether the responses are at a lower or a higher reading level. As our application is targeting a youth demographic, utilizing the categorization system used in grade level school systems will help illustrate whether the training data is biased towards a specific set of reading levels.

Narrative Features

Literacy Insight

As has been previously mentioned, non-mental-health related narrative features in the training data dominate. The combined 'dialogue', 'twist', and 'foreshadowing' narrative types amount to ~75% of the full training data. This underlines the importance of literacy via storytelling elements within the TinyStories dataset over specific outcomes such as 'badending' 'conflict' or 'moralvalue'.

```
1 plot_topic_counts(narrative_probabilities_df, title='Narratives by Type', xlabel='Narrative Type', ylabel='Probability', pre
```



Statistical Insight

Looking at the cumulative probability (~75%) of the 'dialogue', 'twist', and 'foreshadowing' narrative types does not accurately describe the influence of variance due to literacy-related narratives. As we can see by looking at the individual probabilities for these narratives, dialogue accounts for 48% of prompts, and the second-highest is twist with ~18% of prompts. Lastly, foreshadowing only accounts for ~8%. **The ratio is thus about 6:2:1 for dialogue, twist, and foreshadowing in terms of importance. This makes 'dialogue' the strength of the TinyStories dataset, and shows how literacy could be improved by adding more narrative elements such as 'twist' and 'foreshadowing'.**

```
1 print(f"Probability of Dialogue, Twist, and Foreshadowing Narrative: {dialogue_twist_foreshadowing_narrative_probability}")
```



Probability of Dialogue, Twist, and Foreshadowing Narrative: 0.7446579465063359

```
1 # individual probabilities for dialogue, twist, and foreshadowing
2 feature_occurrence_probabilities.loc[['dialogue', 'twist', 'foreshadowing']]
```



feature_occurrences	
feature	
dialogue	0.484367
twist	0.177679
foreshadowing	0.082613

dtype: float64

Key Words

```
1 literacy_levels = [
2     "Kindergarten (5-6)",
3     "1st Grade (6-7)",
4     "2nd Grade (7-8)",
5     "3rd Grade (8-9)",
6     "4th Grade (9-10)",
7     "5th Grade (10-11)",
8     "6th Grade (11-12)",
9     "7th Grade (12-13)",
10    "8th Grade (13-14)",
11    "9th Grade (14-15)",
12    "10th Grade (15-16)",
13    "11th Grade (16-17)",
14    "12th Grade (17-18)"
15 ]
```

```

1 def classify_literacy(literacy_levels, topic_df_words_counts):
2     """ this takes a preset list of literacy levels,
3         and classifies the topic_ids by using the data in the topic_df_words
4         note: there are 12 topic_ids and 13 corresponding literacy levels.
5     """
6     prompt_template = (
7         f"The following is a list of education levels and corresponding ages: {literacy_levels}",
8         f"Use this list of words and their probabilities to classify the most likely level for this set of words: {topic_df_
9         \"Note: Each topic in either list can be assigned only once. Only respond with the topic chosen and nothing else.\",
10        \"Nearest literacy level: \"
11    )
12    prompt = \"\\n\".join(prompt_template)
13    response = client.chat.completions.create(
14        messages=[
15            {\"role\": \"system\", \"content\": \"You are comfortable classifying the language used in stories by simply inferring
16            {\"role\": \"user\", \"content\": prompt}
17        ],
18        model = model,
19        temperature = 0.0
20    )
21    return response.choices[0].message.content

1 # Step 3: creating a word cloud for all topics within a single iteration
2 # add the cluster labels to the batch
3 sample_batch = batches[0]
4 batch_df = pd.DataFrame({'words': sample_batch, 'topic': topic_assignment_batches[0]})
5 # Determine the number of rows and columns for the grid
6 num_topics = len(batch_df['topic'].unique())
7 num_cols = 3 # Adjust as needed
8 num_rows = (num_topics + num_cols - 1) // num_cols
9
10 # Create the figure and axes for the grid
11 fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5 * num_rows))
12 i = 0
13 for topic_id in batch_df['topic'].unique():
14     topic_df = batch_df[batch_df['topic'] == topic_id].copy()
15
16     # Create a word cloud of the topics
17     wordcloud = WordCloud(width=800, height=400, background_color='white', max_words=100)
18     topic_df['words'] = topic_df['words'].apply(lambda x: ' '.join(x))
19     # Concatenate all word strings in topic_df['words']
20     wordcloud_text = ' '.join(topic_df['words'].values)
21     # Filter counts of all words in the wordcloud_text and then select top 100 with count > 1
22     wordcloud_text_counts = wordcloud.process_text(wordcloud_text)
23     wordcloud_text_counts = {k: v for k, v in sorted(wordcloud_text_counts.items(), key=lambda item: item[1], reverse=True)}
24     wordcloud_text_counts = dict(list(wordcloud_text_counts.items())[:100])
25     wordcloud_text_counts = pd.Series(wordcloud_text_counts, index=wordcloud_text_counts.keys())
26     # Filter to words with counts > 1
27     wordcloud_text_counts = wordcloud_text_counts[wordcloud_text_counts > 1]
28     wordcloud_text_filtered = ' '.join(wordcloud_text_counts.index)
29     # Classifying the topic_df['words']
30     topic = classify_literacy(literacy_levels, wordcloud_text_counts.to_string())
31     topic_df['topic'] = topic
32     wordcloud.generate(wordcloud_text_filtered)
33     # Plot the wordcloud on the corresponding subplot
34     row = i // num_cols
35     col = i % num_cols
36     ax = axes[row, col] # Get the current subplot
37     ax.imshow(wordcloud, interpolation='bilinear')
38     ax.axis('off')
39     ax.set_title(f'Topic: {topic}') # Add a title/label
40     i += 1

```