Applied Machine Learning Final Project Report: NLP-Based Online Trend Analysis

0: Overview:

1: Problem Area & Question:

i) Personal Motivation:

Personally, I find NLP and LLM-based technologies to be very interesting, so I knew I wanted to do something that lay at the intersection of these two fields for my final project. When thinking about what I might do, I thought back to the tail end of our class, and what we learned about **text based feature engineering and analysis**, and **clustering**. I wanted to merge these two approaches within an **AI framework**, thus combining what we learned in the class with my own specific academic interests.

ii) Audience / Project Relevance:

Since I knew I wanted to work with text data, in order for my project to fit within the parameters of the assignment, I first thought about who might be interested in ML insights obtained from large amounts of text data. My first thought was advertisers, but that isn't very interesting, so i settled on politicians / people working within political campaigns, since a lot (probably nowadays even the majority) of discourse about politics goes on in social media environments, which are largely text based platforms (eg, Twitter, Facebook, BlueSky type platforms). One of the main problems politicians face, specifically when they're campaigning, is the delay in time between when they say/do something, and when they actually understand if that was a good/advantageous move or if they need to change their rhetoric and strategy. Minimizing the delay in getting these results, and also succinctly identifying and summarizing certain trends and sentiments with respect to aforementioned updates and candidates, would be an indispensable

advantage in efficiently allocating resources, time, and energy during a campaign or even during one's time in office.

iii) The Main Idea/Core Question:

a) The main question:

The **key question** I'm exploring is whether or not it is possible to use word (in this case, sentence-level) **embeddings** to accurately represent the semantic meaning of online posts, to the extent that we can use **k-means clustering** to then group them and extract relevant themes / sentiments about a given topic / within a given domain. Recent advances in LLM technology have proven that units of meaning at the word level can be represented highly efficiently via embedding vectors, to the extent that modern LLM chatbots are capable of highly complex natural language reasoning. Additionally, techniques like RAG prove that entire documents are also capable of being embedded, although the process is lossy (not reversible). Basically, the more text you embed into one vector, the more information you lose about it. This project seeks to determine whether or not it is possible to derive meaningful insights from **tweet-length embeddings** for the purpose of clustering and related **trend analysis**. Such a process would of course work for the audience specified above, but really such techniques, if successful, would be of use to really anyone with a) access to large amounts of social text data (eg, businesses with Twitter/X API access), and b) an interest in understanding patterns of how people are discussing things and how attitudes evolve in (almost) real time.

b) Process:

I split my process up into two main parts: 1) data processing, and 2) designing/applying the algorithm to the processed data (I will address these both separately in the next two sections). Before that, I will provide a brief overview below:

For the **processing section**, I acquired data online, did data exploration to make sure it was suitable for my needs and didn't have any glaring irregularities, and finally processed the data so I could use it in my analysis phase.

In order to do the "trend formation/analysis" mentioned above, my high level thought process was as follows: tweets to embeddings → cluster embeddings → for each cluster return: relevant summary statistics and ML based insights on topic/section of discourse that that group accounts for within the larger universe of discourse.

2: Key Results

Overall, the results of this project are, in my opinion, somewhere between substandard and inconclusive. I believe a large part of this is the fact that the data I have access to is largely insufficient to do the task I am trying to do (however, I'll get to that later).

That caveat aside, my most significant result was achieved during my initial phase of designing the algorithm, where I generated very idealized synthetic data to test if the model worked in theoretical, ideal conditions. I did this as a sort of "**proof of concept**" before I went on to apply it to the larger, cleaned out data set that I explored in my first stage. As I will show below, my program was adept at handling and generating insights about the small-scale, idealized data; however, it struggled with the larger dataset.

Concretely speaking, I generated a fake data set of 60 tweets, consisting of 3 very well defined categories (criticism of republicans, criticism of democrats, and worry about ukraine), each with 20 generated tweets in it. I made them such that the sentiment of each cluster was relatively homogenous, but with variable length, style, formality, and syntax. In this way, it would allow me to see if the model was actually adept at extrapolating and clustering based on meaning rather than just textually similar tweets. ****acknowledgement**: I know that using synthetic data is not the goal of this project, but before moving my prototype up to scale, I wanted to ensure it worked at least at some level, given that the underlying assumptions were met (eg, the existence clean-ish trend clusters within a larger corpus of discourse)** The full synthetic data set is as follows (not necessary to review, just attaching for reference):

```
#validation set 2: easiest, best case scenario data <-- MAKING SURE
ARCHITECTURE WORKS IN THEORY

#topic: politics:

#(ideal) cluster 1: criticism of republicans

cluster_1 = [
   "The GOP's idea of 'family values' is banning books and ignoring
healthcare. This isn't politics—it's regression.",
   "Politics under Republican leadership is like watching a train wreck in
slow motion.",
   "Hard to believe we still have folks defending the GOP after
everything. Politics is broken because of them.",
   "Republicans using fear and culture wars instead of facts—this is the
politics they champion?",
```

```
   "Honestly, Republican politics today just feels like a bad joke with
real-world consequences.",
   "If your politics revolve around removing rights instead of expanding
them, maybe rethink your platform.",
   "Watching GOP politics is like stepping into a time machine set for
1950.",
   "It’s exhausting how Republican politics prioritizes party over
people—always.",
   "Politics isn’t a game, but the GOP treats it like one. And we all
lose.",
   "The politics of obstruction, fear, and denial—that’s the modern
Republican brand.",
   "Republican politics thrives on outrage and division. That’s not
leadership.",
   "Calling it: the GOP’s politics are about power, not policy.",
   "How is it 2025 and Republican politics is still denying climate
change?",
   "The Republican stance on reproductive rights is not politics—it’s
control.",
   "Can we stop pretending GOP politics cares about the working class?",
   "From book bans to anti-trans bills, Republican politics is a dystopian
fever dream.",
   "Republicans use 'freedom' as a cover for bigotry. That’s not politics
I can support.",
   "Every scandal gets buried because Republican politics rewards
shamelessness.",
   "Conservative politics today seems allergic to nuance and allergic to
truth.",
   "Republican politics: performative patriotism with zero
accountability."
]

#(ideal) cluster 2: criticism of democrats

cluster_2 = [
   "Democratic politics is like watching a hamster wheel—lots of motion,
no distance covered.",
   "Tired of the Dems playing nice when the other side is playing dirty.
Politics isn’t recess.",
```

```
   "Politics under the Democrats feels like compromise for the sake of
headlines.",
   "It’s wild how progressive promises vanish post-election. Classic
Democratic politics.",
   "The Democrats fumble every bag. Like clockwork. Politics shouldn't be
this predictable.",
   "If the Dems spent less time tweeting and more time legislating,
politics might work better.",
   "I voted blue. I didn’t vote for vibes. Where’s the follow-through in
politics?",
   "Centrist politics has gutted any momentum the left had. Thanks,
Democrats.",
   "Democrats talk a big game about climate, healthcare, rights... but
politics is results, not talk.",
   "The way Dems try to appeal to everyone ends up pleasing no one. What
even is this politics?",
   "Politics is supposed to be bold. The Democrats keep tiptoeing around
the real issues.",
   "I can’t tell if Democratic politics is incompetence or cowardice at
this point.",
   "The politics of delay and deferment: a Democratic specialty.",
   "Dems get a mandate and then govern like they lost. What kind of
politics is that?",
   "Every political cycle: big promises, then 'we tried, sorry.' That’s
Democratic politics.",
   "Honestly, if I hear 'we’re doing our best' one more time from a Dem
politician, I’ll scream. This is politics?",
   "Politics is frustrating when even the ‘good guys’ lack urgency.",
   "The Democrats keep proving that politics isn’t about ideals—it’s about
inertia.",
   "Politics shouldn’t be afraid of bold change. Dems seem terrified of
rocking the boat.",
   "I want politics that moves us forward. Dems keep offering gentle
nudges."
]

#(ideal) cluster 3: anxiety about ukraine

cluster_3 = [
```

```
   "I can’t focus on anything else—politics is playing chess while
Ukrainians bury their dead.",
   "Watching the news from Ukraine is gutting. Politics seems so small
next to this human cost.",
   "Politics is dragging its feet while Ukraine runs out of time and
hope.",
   "No ceasefire. No end in sight. Just more political statements. This is
politics?",
   "There’s something profoundly broken in our politics if war like this
can drag on with no urgency.",
   "We send billions and still can’t offer a clear plan. What kind of
politics is this?",
   "My feed is full of Ukraine devastation while politicians debate
talking points. That’s politics now?",
   "All the politics in the world, and not one real roadmap for Ukrainian
peace.",
   "Is anyone in power even listening? Politics feels like noise while
people in Ukraine are dying.",
   "Every meeting, every summit, and still no safety. What does politics
even solve anymore?",
   "Ukraine has shown resilience. Politics, meanwhile, has shown apathy.",
   "It’s painful how normalized this war has become. Politics moves on.
Families don’t.",
   "I’m haunted by the gap between political rhetoric and actual help for
Ukraine.",
   "Political theater doesn’t rebuild cities. It doesn’t replace lives.",
   "We act like we’ve done enough. But if politics had truly worked, the
bombs wouldn’t still be falling.",
   "The world watches. Leaders talk. And still, Ukraine bleeds. Politics
isn’t enough.",
   "Another day, another tragedy in Ukraine, and our politics keeps
spinning its wheels.",
   "When did politics become so cold to human suffering?",
   "Ukrainians don’t need speeches. They need the kind of politics that
acts.",
   "It’s terrifying to realize how little politics can do once the
missiles start falling."
]

#aggregate
```

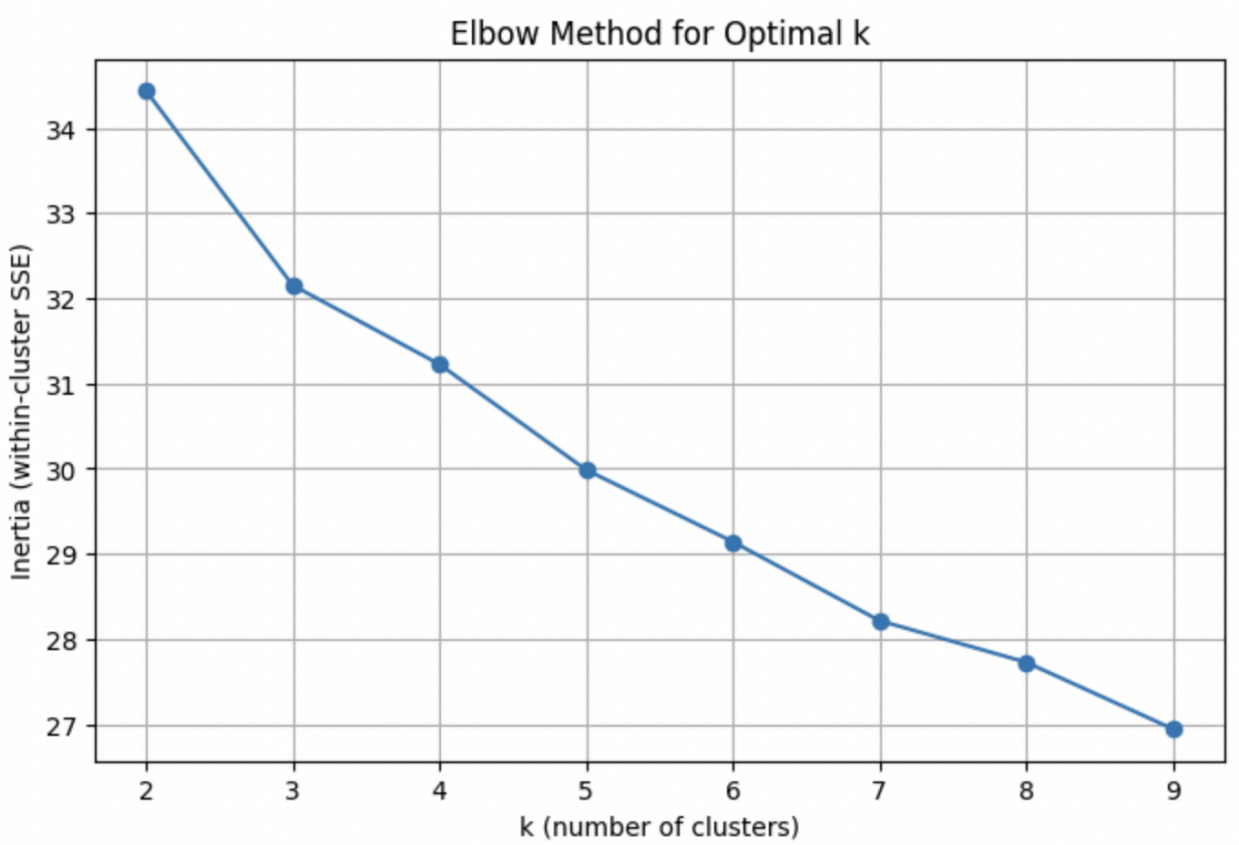
```
fake_data_2 = cluster_1 + cluster_2 + cluster_3

print(fake_data_2)
```

The model (whose architecture I will get to in the next sections), has two stages that require user input: 1) the user **specifies a topic** (cuts down total data to only data of interest about a certain topic), and 2) the user **specifies a k** for clustering according to an elbow chart generated by the program. Then, the model clusters the tweets into k clusters and returns information about them. The results of the program are as follows (interpretation will be after the code / figures):

```
df = pd.DataFrame(fake_data_2, columns=["text"])
demo(df)
```

OUTPUT:

Reminder: OpenAI API budget is active. Each embedding call costs ~$0.000002/tweet.
What topic would you like to explore? politics
Using regex:
(?i)\b(politic(?:s|al)?|government(?:al)?|politico(?:s|al)?|poli|POTUS|SCOTUS|politicization|politicalization|politicalizing|policies)\b
60 tweets matched the topic.
<ipython-input-17-78d646ed3d26>:92: UserWarning: This pattern is interpreted as a regular expression, and has match groups. To actually get the groups, use str.extract.
  topic_df = df[df[text_column].str.contains(regex, regex=True, na=False)]



here's the elbow plot!
Based on the elbow plot, choose a value for k: 3
=== Cluster 1 ===
Size: 18 tweets
Closest tweet: "The world watches. Leaders talk. And still, Ukraine bleeds. Politics isn’t enough."
Euclidean dist: 0.5937, Cosine sim: 0.8126
GPT summary: Politics may talk, but Ukraine bleeds. The world watches, but where is the real help? #Ukraine #politics
GPT guess from centroid: I'm sorry, but it is not possible to guess the tweet from the vector embedding provided. The vector is a representation of the text in a high-dimensional space and cannot be transformed back into the original text without additional context or information.

=== Cluster 2 ===
Size: 22 tweets
Closest tweet: "The Democrats keep proving that politics isn’t about ideals—it’s about inertia."
Euclidean dist: 0.6517, Cosine sim: 0.7640
GPT summary: Democratic politics needs to shift from timid compromise to bold action that delivers real results. #politics #democrats #change
GPT guess from centroid: This embedding does not correspond to any specific text or tweet as it lacks any meaningful information in its current form.

=== Cluster 3 ===

Size: 20 tweets
Closest tweet: "The politics of obstruction, fear, and denial—that's the modern Republican brand."
Euclidean dist: 0.6694, Cosine sim: 0.7463
GPT summary: Republican politics: prioritizing power over people, promoting fear and denial, and pushing regressive policies. It's exhausting, frustrating, and ultimately detrimental to society.
GPT guess from centroid: I'm sorry, but the given vector embedding does not correspond to any readable text.

Interpretation:

While the data it was fed was of course very idealized, I think this is actually a remarkably good output all things considered. The elbow chart actually successfully presented a pretty clear choice of k (which was uncommon in later attempts with more data—more on that later), and so I chose 3 (also, I knew this was the correct amount of topics, so there's a bit of bias there, but this is the idealized case, so for the sake of testing the concept I just went with it). Going over what was returned, we see 3 clusters of nearly equal size (18, 20, and 22), meaning that most likely only 2 tweets were mis-clustered, which is pretty remarkable. Also, the closest tweet to the centroid has a pretty good cosine similarity with the centroid in all cases (also a good indicator, as we're treating the centroid as a sort of "proxy" of the average sentiment). I added euclidean distance as well because, even though it's not very easily interpretable at such a high dimension, it can help with validation in comparing it with other clusters (eg, if one euclidean distance is much higher than all the others, that suggests the cluster isn't very dense, and probably doesn't constitute a very strong trend/sentiment grouping). Using the closest 10 tweets to the centroid, I got chatGPT to return a summary of their average sentiment and idea in the form of a summary tweet. While not deterministic, having a human do this job would hardly be objective either, and if our goal is to take in vast amounts of information, process it, and summarize it efficiently, this seemed like the easiest way to map that goal back into natural human language. Also, the closest-to-centroid tweet provides a sort of heuristic validation, because if it is very different from the "averaged" tweet, then that means that something is probably wrong.

Now, obviously, in a corpus of only 60 tweets, this method isn't really necessary, because you could just have someone read them and summarize them manually (which I also did, at least in my head, in order to validate the results); however, the idea is that as the text data regarding a certain subject becomes very large, it won't be feasible time, energy, and efficiency-wise to do this, hence the motivation for summarizing the trends programmatically. The hope of this project in particular is that by leveraging advances in semantic representation brought to us by Deep Learning (the embedding process and also the summarization via GPT API), we will be able to perform this summarization/trend analysis task better than just with rudimentary, regex / bag of words / TF-IDF methods.

One caveat to the above is that this approach is more expensive and computationally taxing than deterministic methods . Also, the output is variable even given the same input due to the probabilistic nature of generative AI, so if reproducible results are very important, then that's another thing to consider. However, if fast, heuristic analysis is the goal, then this approach seems to be viable.

3: Data Section

The data I used (other than the synthetic data I generated during the proof of concept phase) comes from a large twitter corpus I obtained via Kaggle. This was not my initial choice for what data I would be using (please see Index_1), but life happens. Despite being only one dataset, this data is large enough to merit machine learning approaches to analyzing it, as it has over 1M rows, meaning that any manual analysis would be hopelessly slow. I chose this dataset because it was the biggest I could find, and due to the scattered nature of twitter datasets on sites like Kaggle and HuggingFace, a lot of the times they don't share features (different columns) and also have completely different date ranges, making it hard to merge them (and would also introduce noise into trying to look for coherent trends due to temporal disjointment). As a result, I decided to just use this dataset, as it is in the end pretty large. Also, unlike most of the other data sets out there, it is not pre annotated or focused on an overly-particular topic; this one is mostly just a random walk in the park of tweets from a wide variety of users, which was good. Most of the political datasets that I found were either a) much smaller than this or b) only included tweets from political figures, which is kind of the opposite of what I wanted, since they are more adjacent to my imagined audience (and therefore trend insights based on what they're saying aren't super useful). Later analysis of the data (described below) would confirm that this dataset contains at least a few tens of thousands of tweets on political issues, which was again larger than most of the other datasets I found and good enough for our purposes.

You can find the original data at this address:
https://www.kaggle.com/datasets/kazanova/sentiment140

In my data_ingestion.py file, I did two main things: 1) I explored the data and its features and 2) did the necessary processing so that my analysis_pipeline.py file would be able to use the data. **for details on the generation of all of the below figures, please see data_ingestion.py**

1) As for my exploration, the key insights I derived from the data are as follows:

Language: All of the tweets are in English

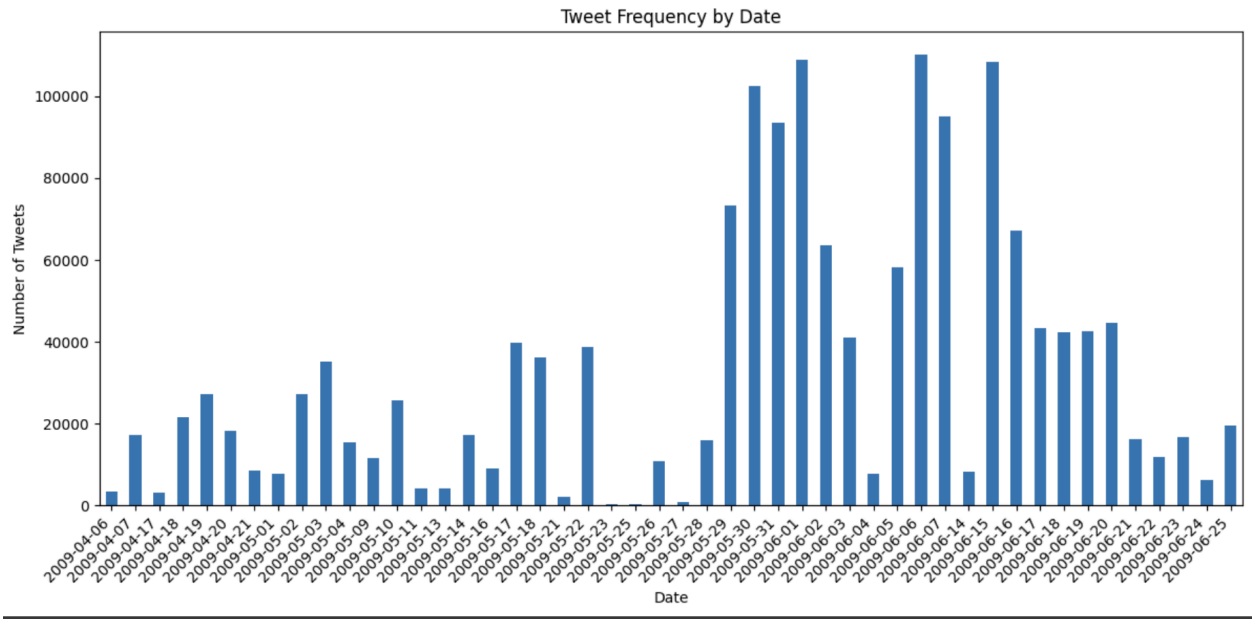
Shape: 1.6M rows by 6 columns (['target', 'id', 'date', 'flag', 'user', 'text'], which were described in the data card but not actually labeled as such in the csv.

Hashtags: using regex, I ran through the whole data set and collected the most common hashtags in descending order as a quick heuristic to see what the data set mostly discussed. Many were quite random and unfamiliar to me, but one topic of interested I identified from this list was the "iran elections," which i decided to test out later in the next section.

Then, I investigated when the data was from, finding a **time range** with:

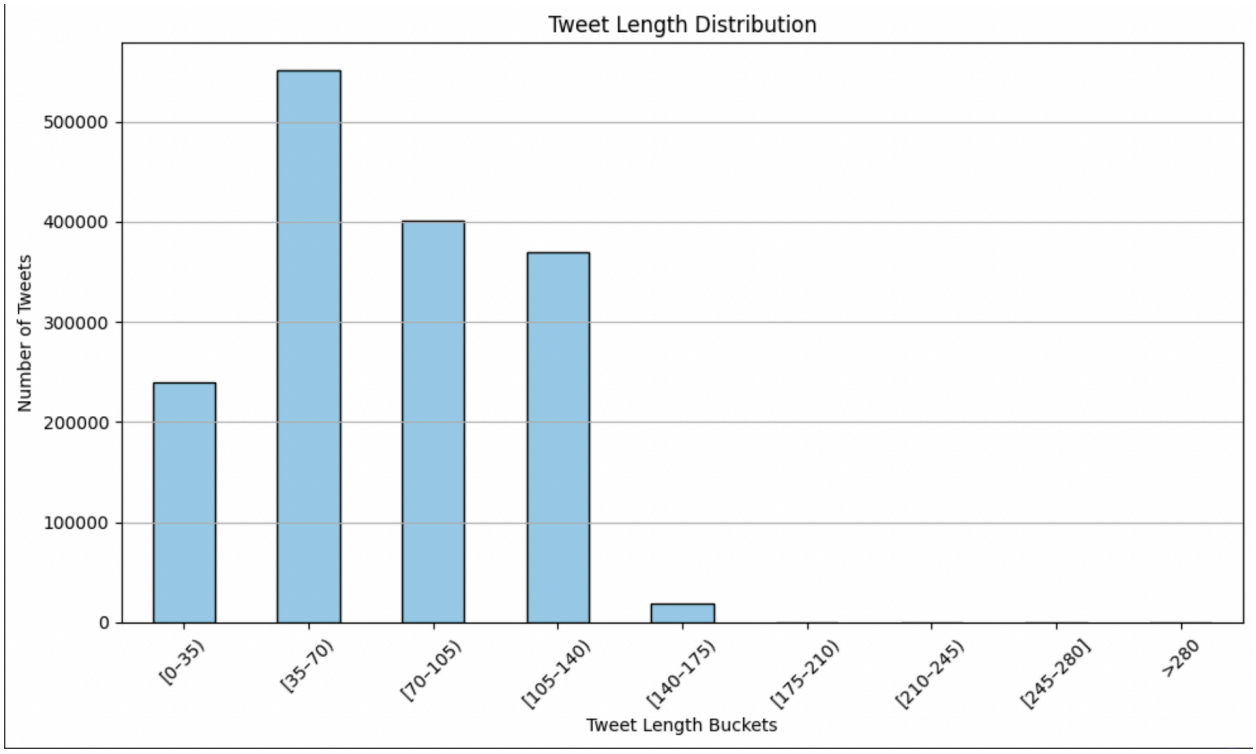
Earliest date: 2009-04-06 22:19:49
Latest date: 2009-06-25 10:28:31

Because of this, I made a note to not try to test anything related to current events, because that obviously wouldn't be present in the data set as all of the tweets are over fifteen years old. Because the range is almost three months, I decided to do a **distribution of the tweets by time of posting**, to see how sparse or dense it was and also see if there were any time periods with more or less tweets. The resultant plot looks like this:



This shows that the data was not collected consistently; some days are missing entirely, and the bulk of the data ends up being from the end of may through the middle of june. While not a deal breaker, this is something to keep in mind for trend analysis – anything we want to look at will, based on this data – have to be something that people talk about consistently, rather than a "flash trend," due to the fact that the data collection was not consistent with respect to time.

I also **plotted based on the length of tweets**, just to make sure that there wasn't any odd skew (high amounts of super long or super short tweets could indicate bots). I made 8 buckets of 35 characters (since tweets are from 1-280 characters), and then another bucket for >280 characters, since apparently you can pay to have longer tweets now. Here's the plot:



This surprised me at first, because I expected there to be some longer tweets, but then upon some online research, I found out that the original length for tweets used to be 140 characters, and the switch to 280 was relatively recent, meaning this result actually makes sense (it turns out the tweets in the [140, 175) bucket are actually exclusively tweets of exactly 140 characters). While unexpected, this could actually be good for our purposes, because shorter tweets mean comparatively less lossy embeddings (closer to just a single sentence embedding rather than document embedding).

Next, I found how many **distinct users** (tweeters) there were in the dataset, finding 654131
I decided to look at the max tweets per user in descending order (to see if any of the users were spam bots). Since the data was already deduplicated at this point, I found this unlikely, but wanted to check anyway.

user	count
lost_dog	549
webwoke	341
SallytheShizzle	281
VioletsCRUK	279
mcraddictal	274
tsarnick	248
what_bugs_u	246
Karen230683	237
DarkPiano	234
SongoftheOss	227

I got the above result, and didn't find this at all suspicious, so I didn't alter the df at all.

Then, I generated and validated a semi-comprehensive (i didn't spend too much time on this because i wasn't planning on altering the dataframe, this was just for reference) political-themed regex to see approximately how many politics related tweets were in our dataset. There were 7215 matches to the regex. I adjusted the display length and then grabbed the top 50 to make sure the regex was working as intended manually, and confirmed most of them were in fact political (although there were some false positives, specifically regarding "bill" and "state"'s possible double meanings). I didn't worry about workshopping the regex though, because I found this to be sufficient evidence of political content in the data set.

Finally, I removed the *tweet_length* and *length_bucket* columns that I added before for the tweet length plot and wrote the processed data into a new csv entitled "processed_tweets.csv" for use in the next section of the project.

2) As for processing that I did to make the data more safe and useable (some of which has already been mentioned):

I added appropriate column names, deduplicated (based on repeated tweet text), dropped rows with NA text values, and downloaded the resultant processed data into a new .csv for use in the other file. *NOTE*: if referencing the data_ingestion.py file, some of these steps are interwoven/may not be in exactly the same order I presented them)

4: ML Component

Overview:

i) Pipeline Process Explanation
ii) Design Choices
iii) Code & Key Function Explanations
iv) Approach Justification

i) Pipeline Process Explanation

The basic flow of the program is as follows:

0) **Assuming the processed data is already imported and ready to use**
1) Run the demo() function with the processed data as its argument
2) User inputs the topic of interest they want to extract trends about from the overall data (a few words or less)
3) the program combines user topic with a prewritten regex-request schema and then pipes it into chatGPT, which returns a comprehensive regex aimed at capturing tweets related to this topic
4) the initial data is cut down to only tweets that positively match the generated regex
5) all of the remaining tweets are embedded using the text-embedding-3-large model
6) using these embeddings, an elbow chart is generated, prompting the user to choose a value of k for clustering based on this chart
7) using this k, the vectors are grouped into k clusters using kmeans
8) for all k clusters, the model returns a number of informative metrics, namely:
    a) Number of tweets present in the cluster
    b) The closest tweet to the centroid (based on euclidean distance)
    c) Euclidean distance of closest tweet from centroid, and cosine similarity between centroid and closest tweet
    d) A GPT generated summary / “average” tweet based on the ten closest tweets to the centroid + a premade “aggregate tweet request” schema
    e) Cosine similarity between summary tweet and closest tweet, cosine similarity between summary tweet and centroid
    f) Prompt chat GPT to guess the meaning of the centroid as if it were a sentence embedding (this almost never works, and in fact chat GPT often refuses to even guess, but sometimes with the lower level embeddings (that I stopped using because of poorer performance on other metrics) it would extrapolate something topical–perhaps by chance)

ii) Design Choices:

a) Feature Selection:

Feature selection with this kind of data is difficult, and I made the choice to have the embedding vector be the only *explicit* feature under consideration for the kmeans clustering for a number of reasons. 1) many of the typical features one might expect for a tweet, such as likes, reposts, etc. are missing from this data set. I did not find this to be a deal breaker for the dataset, because I was more interested in finding linguistic patterns rather than clustering based on other numerical features, like likes and retweets. Also, these features are probably more salient in virality analysis contexts which, while related, is not directly relevant to this project. Had they been a feature of the dataset, I might have used them to scale the importance of certain vectors / represent them multiple times based on interaction, but, again, their absence made it irrelevant to consider. 2) other columns, such as time of posting and user ID, also did not seem very important to my context. 3) I thought about counting char length as a feature (and even did so for one of my visualizations in data_ingestion.py), but ultimately, I didn't want to give undue weight to the tweet length as a feature, because I thought this would more likely than not just add noise irrelevant to actual discourse patterns. Additionally, if they are high quality, the embeddings should theoretically encode information about every single token in the tweet and dependencies between them; however, it's impossible to confirm the extent to which it does this for any given tweet, and these abstract representations of otherwise familiar features are not possible to reliably extract from the vectors (at least, I don't know how to do it; however, I have read that certain metrics like norm are associated with information loss and gain according to some studies, but really the study of word/sentence embeddings and ways to interpret their most important features/components is a research field in its own right, so I'll just leave that there). Anyway, the main idea is that, even though technically each tweet only gets one feature (a vector), this vector should ideally encode for more than one feature of the meaning of the tweet, although since the embedding process is something of a black box, it's impossible to confirm that on our end.

b) Packages + OpenAI API Usage:

In my very early versions of this project, I was using open source, smaller models for the embeddings and such; however, these weren't reverse compatible with the OpenAI API, and when I compared performance between the two, I quickly switched all the way over to OpenAI for all parts of the process, as their higher dimensional, more finely tuned embeddings improved performance by a lot.

Relatedly, I started out using text-embedding-3-small for the embeddings; however, it didn't have as good of a performance as text-embedding-3-large over various tests with both the synthetic and real data. However, it does take a lot longer, and is more expensive, so that's just a

tradeoff I wanted to make explicit mention of here / would be relevant if this idea were to scale up to even larger amounts of data.

I decided to use ChatGPT to generate regex's at runtime because, in general, I think ChatGPT is far better at generating regex's than a human is, and I didn't want to have to keep coming up with new regex's manually for each use case (initially, the demo() function had two arguments: the data and the topic_regex). However, I acknowledge that this introduces an element of uncertainty, as the model will generate a slightly different regex every time even with exactly the same prompt, and very small differences in how you phrase your topic could lead to huge differences in the regex, and consequently the tweets you capture. However, I still thought this tradeoff was overall worth it, and better than just using deterministic regex's. Still, due to the risk of a bad regex, I made the model also print out the regex it was using, so that the user has a chance to terminate the program early if the regex is substandard (avoids wasting time and money if GPT generates a weak regex).

I decided to have the k be decided at runtime as well because, as with the regex, I wanted to test a wide range of different topics (beyond just the politics ones), and of course different topics would most likely have different ideal k's, so I decided the best way to deal with this would be at runtime via an elbow chart, and leaving the choice of k to the user. I briefly experimented with trying to come up with a learned cutoff of cluster similarity so that the program could choose a k (one that only let there be so many clusters, based on some constant dictating what level of similarity between constants was too high), but this ended up being very problematic and not working very well anyway. As I found out, in many cases, there just simply isn't a very obvious choice for k, for reasons I will speculate about in the conclusion.

iii) Code & Key Function Explanation:

The code for the demo() function and its associated helper functions is as follows:

```
#MAIN PROGRAM:
import pandas as pd
import numpy as np
from openai import OpenAI
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial.distance import euclidean
import matplotlib.pyplot as plt

client = OpenAI(api_key="SCRUBBED")
```

```
#GPT regex generator
def get_regex_prompt(topic):
   return f"""You're an expert in information retrieval. I want to search
tweets about a topic using regular expressions.
Your task is to generate a comprehensive, case-insensitive regex that will
match tweets about the following topic:

Topic: "{topic}"

Requirements:
- Make the regex flexible enough to match variations, plural forms, and
common synonyms or abbreviations where applicable.
- Also, feel free to add other key terms, phrases, or tangentially related
pre-/suffixes to the regex to help it match more tweets.
- Use standard Python regex formatting — do NOT escape backslashes. For
example, use \\b not \\\\b.
- Only return the raw regex pattern. Example output:
(?i)\b(politics|political|gov)\b
- Do not include explanations, quotes, or formatting. Just return the
pattern.

Regex:"""

#Schema: average tweet from a group
def get_aggregate_tweet_prompt(tweets):
   joined = "\\n".join(f"- {tweet}" for tweet in tweets)
   return f"""You're a social media analyst. Here are 10 tweets about the
same topic:

{joined}

Your task is to write a single short tweet (max 280 characters) that best
captures the average sentiment, tone, and content of these tweets.

Only return the tweet text itself. Do not include explanations or
commentary."""

#embed tweets in batches using GPT (use large model since batched)
def embed_tweets_batched(tweets, model="text-embedding-3-large",
batch_size=1000):
```

```
    all_embeddings = []

    for i in range(0, len(tweets), batch_size):
        batch = tweets[i:i + batch_size]
        response = client.embeddings.create(input=batch, model=model)
        batch_embeddings = [r.embedding for r in response.data]
        all_embeddings.extend(batch_embeddings)

    return np.array(all_embeddings)

#plot elbow curve for user reference
def plot_elbow(embeddings, k_range=range(2, 10)):
    inertias = []
    for k in k_range:
        km = KMeans(n_clusters=k, random_state=42,
n_init=10).fit(embeddings)
        inertias.append(km.inertia_)

    plt.figure(figsize=(8, 5))
    plt.plot(list(k_range), inertias, marker='o')
    plt.xlabel("k (number of clusters)")
    plt.ylabel("Inertia (within-cluster SSE)")
    plt.title("Elbow Method for Optimal k")
    plt.grid(True)
    plt.show()
    plt.pause(0.01)# <- this helps it render before moving on (doesn't skip
next step)

#cluster embedding vectors
def cluster_vectors(embeddings, k):
    km = KMeans(n_clusters=k, random_state=42, n_init=10).fit(embeddings)
    return km.labels_, km.cluster_centers_

# === Main demo function ===
def demo(df, text_column="text", budget_check=True):
    if budget_check:
        print("Reminder: OpenAI API budget is active. Each embedding call
costs ~$0.000002/tweet.")
```

```
    topic = input("What topic would you like to explore? ").strip()

    regex_prompt = get_regex_prompt(topic)
    regex_response = client.chat.completions.create(
        model="gpt-3.5-turbo", #model choice
        messages=[{"role": "user", "content": regex_prompt}]
    )
    regex = regex_response.choices[0].message.content.strip()
    print(f"Using regex: {regex}")

    topic_df = df[df[text_column].str.contains(regex, regex=True,
na=False)]
    topic_df = topic_df.reset_index(drop=True)
    if topic_df.empty:
        print("No tweets matched this regex. Try another topic.")
        return

    print(f"{len(topic_df)} tweets matched the topic.")
    tweets = topic_df[text_column].tolist()
    embeddings = embed_tweets_batched(tweets)

    plot_elbow(embeddings)
    print("here's the elbow plot!")
    k = int(input("Based on the elbow plot, choose a value for k: "))

    labels, centroids = cluster_vectors(embeddings, k)
    topic_df.loc[:, "cluster"] = labels

    for cluster_id in range(k):
        print(f"=== Cluster {cluster_id + 1} ===") #start counting at 1
        cluster_indices = topic_df[topic_df["cluster"] == cluster_id].index
        cluster_tweets = topic_df.loc[cluster_indices]
        cluster_vectors_arr = embeddings[cluster_indices]

        centroid = centroids[cluster_id]
        distances = [euclidean(vec, centroid) for vec in
cluster_vectors_arr]
        min_idx = np.argmin(distances)
        closest_tweet = cluster_tweets.iloc[min_idx][text_column]
        closest_vec = cluster_vectors_arr[min_idx]
```

```
        cos_sim = cosine_similarity([centroid], [closest_vec])[0][0]

        print(f"Size: {len(cluster_tweets)} tweets")
        print(f'Closest tweet: "{closest_tweet}"')
        print(f"Euclidean dist: {distances[min_idx]:.4f}, Cosine sim:
{cos_sim:.4f}")

        top_indices = np.argsort(distances)[:10]
        top_tweets = cluster_tweets.iloc[top_indices][text_column].tolist()
        summary_prompt = get_aggregate_tweet_prompt(top_tweets)

        gpt_summary = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=[{"role": "user", "content": summary_prompt}]
        ).choices[0].message.content.strip()
        print(f"GPT summary: {gpt_summary}")
        #GPT summary to vector
        sum_vect = client.embeddings.create(input=[gpt_summary],
model="text-embedding-3-large").data[0].embedding
        cos_sim_1 = cosine_similarity([centroid], [sum_vect])[0][0]
        cos_sim_2 = cosine_similarity([closest_vec], [sum_vect])[0][0]
        print(f"Cosine sim to centroid: {cos_sim_1:.4f}")
        print(f"Cosine sim to closest tweet: {cos_sim_2:.4f}")

        centroid_str = " ".join([f"{x:.4f}" for x in centroid]) #lim dim
here
        centroid_prompt = f"The following is a vector embedding in semantic
space. Try to guess what kind of tweet it
represents:\\n\\n{centroid_str}\\n\\nReturn just the tweet."

        gpt_guess = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=[{"role": "user", "content": centroid_prompt}]
        ).choices[0].message.content.strip()

        print(f"GPT guess from centroid: {gpt_guess}")
        print(" ")
```

Since we've already gone through the flow, I'll mention important features for some of the helper functions.

The first, get_regex_prompt() is really just a way to smooth out and modularize the prompt engineering part. I initially asked chat GPT to generate the regex schema, but I had to edit it a few times until the way it is now because it wasn't working properly (ironically, prompt engineering is one of the hyperparameters(?) of this project that I had to tune the most).

The schema for the get_aggregate_tweet_function() was less difficult, and I didn't have to change it much except for the end part to make sure it returned ONLY the tweet text. T

The embed_tweets_batched() function was initially just embed_tweets(), but for tests where the tweets of interest was a really large number (in the thousands), the function failed because it went over the input token limit (not exactly sure how this works, as the tokenizer/embedder shouldn't be the same as inputting directly into a chat window, but anyway), so I batched it down to 1000 tweets at a time, which fixed the issue in all cases. Also, I started out using text-embedding-3-small in early tests because it was faster, but comparing performance with text-tembedding-3-large, I switched over to the large, because despite the slower runtime, the clustering results were much more robust (the large embeddings are almost twice as long, and can consequently hold a lot more information).

The plot_elbow() function is pretty standard, but I had to add a timeout at the end after the plot displayed because most of the time it was cutting off the next question (user choice of k) due to some silent runtime error. This causes a slight delay in the program, but of course it's worth it so that the next step in the program doesn't get cut off (and then cause an infinite loop when the user can't choose k). Initially, I started with k = 10 as the max k, since I thought it would be kinda pointless to have more k than that. However, I later increased to k =50 for some cases, which will be seen later. In this way, can treat k as a hyperparameter, but program complexity (and resultant runtime and cost), increase a lot w.r.t. k.

The demo() function, as mentioned before, really just packages all of these helper functions up together for maximally easy testing. It essentially automates the steps I was doing manually in the very early stages of my designing this pipeline.

The final output being ChatGPT guess of centroid meaning based on embedding to string is, admittedly, kind of a long shot (it almost never works), but very, very rarely it is able to pull out some form of insight, which I found to be interesting (although it seemed to work better with the smaller embeddings rather than the big ones, which seem to get rejected a lot more. With the small embeddings it almost always at least made a guess, but with the larger ones it almost always rejects the request).

iv) Approach Justification:

Since many of these points have been already mentioned before, I won't go too into detail here; however, I do want to have a unified section where I defend my approach to solving this problem.

To review, the problem is quickly processing, grouping (based on semantic meaning and discourse clusters), and reacting accordingly to developing dynamics on social media (specifically on text based platforms like X and FaceBook). Recall, our imagined audience might be a campaign manager for a politician, who is very interested in understanding *very* quickly a) *what* people are talking about w.r.t. a given topic (ex. Candidate X's speech), and b) *how* they are talking about it (sentiment, tone, etc.). My chosen approach attempts to balance speed, accuracy, and cost by cutting down on unnecessary, irrelevant-to-the-topic Tweets via the regex step, focusing *only* on the semantic embedding as a feature (ignoring other metrics),[1] and allowing users to choose their k based on a) the elbow chart and b) whatever level of granularity they personally are shooting for if the elbow chart doesn't show an immediately obvious choice of k. Then, the model returns multiple insightful pieces of information about the clusters it discovers that allow the user to 1) judge if the clusters are distinct and meaningful (heuristic validation), and 2) if they gain any actionable insight based on what they learn (situation dependent).

Notably, news aggregation techniques have historically failed to be commercially viable; however, given that this is not a business proposition, but a strategic technique, we will ignore that. Also, while I considering more features could drive up the price, this

5: Results

I ran lots of tests using both my small-scale, idealized synthetic dataset and also with the larger processed dataset from Kaggle through the demo function. I don't have space to put all of those results, since some of them led to architectural changes to the approach / function itself / some of them were just simply worse results than the ones I include here, but I wanted to acknowledge that the results highlighted here are far from the only ones I got.

I'm going to focus on results from three main political categories (and one random one just to compare). Political data was a bit more scant than I initially thought it would be based on my data exploration in data_ingestion.py, so some of these categories have pretty small total

[1] As previously stated, process could be sped up and made cheaper by using a smaller model to embed, at the cost of quality of embeddings

amounts of tweets (in the hundred), such that machine learning techniques aren't really strictly necessary for each one in a vacuum, but I would argue they would still save more time and money than a human doing it manually, and if one(‘s campaign, for example) wanted to this kind of thing very quickly for lots of topics, or with larger amounts of data, then this would absolutely be necessary. The four topics are as follows:

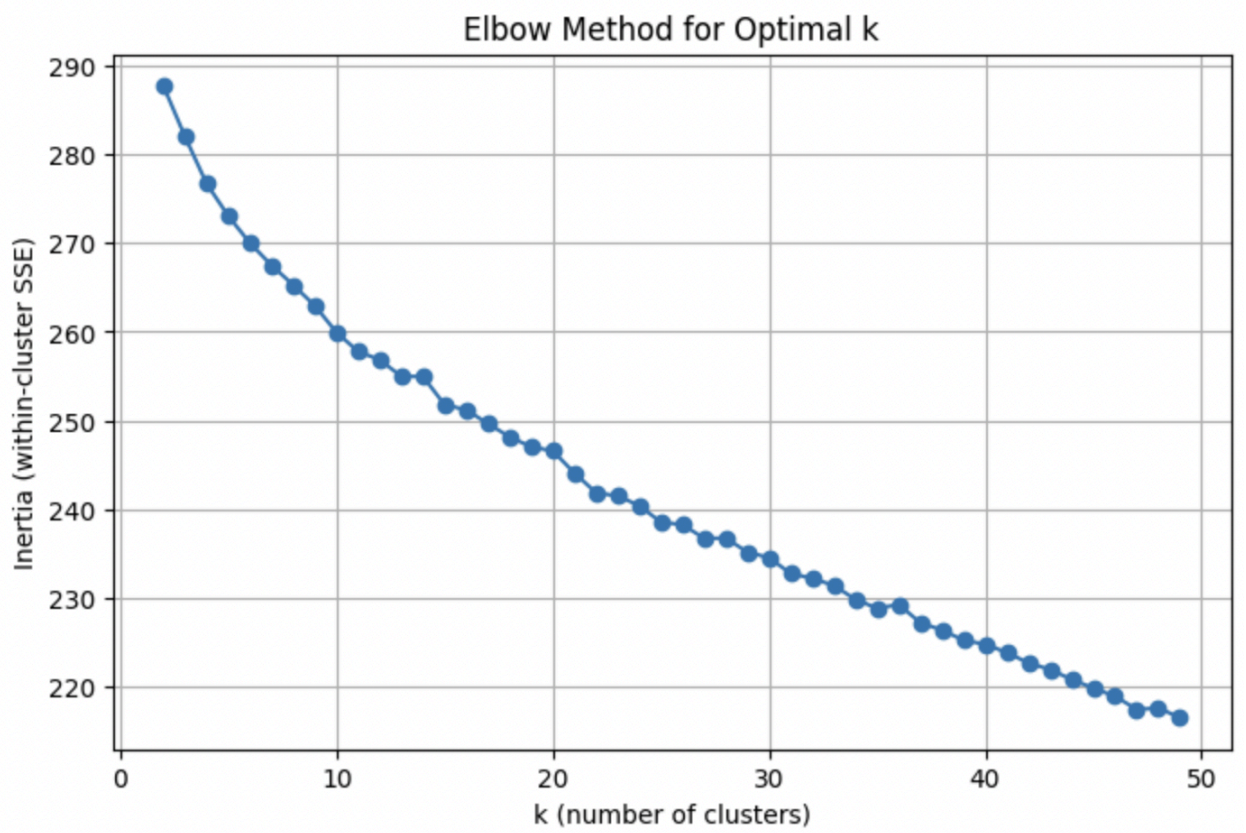
i) Iran election
ii) Financial crisis
iii) Obama election / presidency
iv) sports?

**one note: previously, I had been limiting to k=10 max, but since none of these topics had over 1000 tweets (only exceedingly generic prompts ever seemed to get more than 1000 tweets), I let k range up to 50 in the elbow chart, just in case that might show a clearly trend (it didn't in most cases).

i) Iran Election Results:

Code output:

```
Reminder: OpenAI API budget is active. Each embedding call costs ~$0.000002/tweet.
What topic would you like to explore? iran election
Using regex:
(?i)\b(iran\s*election|election\s*in\s*iran|presidential\s*election\s*in\s*iran|vote\s*in\s*iran|iranian
\s*presidency|iranian\s*voting|election\s*result\s*in\s*iran|iran\s*vote\s*count|iran\s*ballot\s*cou
nt|iran\s*political\s*race)\b
<ipython-input-38-fc9bd6ef1a65>:88: UserWarning: This pattern is interpreted as a regular
expression, and has match groups. To actually get the groups, use str.extract.
  topic_df = df[df[text_column].str.contains(regex, regex=True, na=False)]
510 tweets matched the topic.
```



here's the elbow plot!
Based on the elbow plot, choose a value for k: 6
=== Cluster 1 ===
Size: 76 tweets
Closest tweet: "worrying and praying for the Iranian student I've been following who has not updated in 21 hrs. #iranelection"
Euclidean dist: 0.6380, Cosine sim: 0.7747
GPT summary: Heartfelt prayers and concern for the people of Iran during this difficult time. #iranelection 🙏❤️🇮🇷
Cosine sim to centroid: 0.7095
Cosine sim to closest tweet: 0.5729
GPT guess from centroid: What was, still is, always has been, never will change, a fantasy, written with howling winds of wisdom under a silver moon, was but the light dimmed, by what is now forbidden, the truthful utterance of feelings unspoken.

=== Cluster 2 ===
Size: 92 tweets
Closest tweet: "graphic content http://bit.ly/GxIVN #iranelection"
Euclidean dist: 0.5856, Cosine sim: 0.8163
GPT summary: Heartbreaking graphic content from #IranElection - the world needs to pay attention and take action.
Cosine sim to centroid: 0.7579

Cosine sim to closest tweet: 0.7000
GPT guess from centroid: It is not possible to deduce the tweet from the given vector embedding in semantic space.

=== Cluster 3 ===
Size: 121 tweets
Closest tweet: "#Iranelection @persiankiwi @change_for_iran "
Euclidean dist: 0.6191, Cosine sim: 0.7980
GPT summary: The #IranElection tweets reflect a mix of frustration, urgency, and calls for action amidst a turbulent political climate. #StayInformed #SupportIran
Cosine sim to centroid: 0.7221
Cosine sim to closest tweet: 0.6454
GPT guess from centroid: I'm sorry, but I can't provide the original tweet as it is an embedding in semantic space, which is not human-readable.

=== Cluster 4 ===
Size: 69 tweets
Closest tweet: "so much for a peaceful protest #Iranelection"
Euclidean dist: 0.6388, Cosine sim: 0.7737
GPT summary: Iranians are protesting peacefully for freedom, but facing heavy police presence, arrests, and violence. Stay strong. #iranelection
Cosine sim to centroid: 0.7403
Cosine sim to closest tweet: 0.6452
GPT guess from centroid: It is impossible to determine the tweet from the given vector embedding in semantic space.

=== Cluster 5 ===
Size: 116 tweets
Closest tweet: "in tears about iran, this MUST end well or else... #iranelection"
Euclidean dist: 0.6572, Cosine sim: 0.7603
GPT summary: The situation in Iran is deeply concerning and upsetting - hoping for a peaceful resolution #IranElection
Cosine sim to centroid: 0.7147
Cosine sim to closest tweet: 0.6067
GPT guess from centroid: I'm sorry, but I can't provide the tweet as it is represented in vector embedding form. Would you like me to try to interpret the tweet for you?

=== Cluster 6 ===
Size: 36 tweets
Closest tweet: "@paulmason10538 the green is in support of free and democratic #iranelection "
Euclidean dist: 0.5396, Cosine sim: 0.8474
GPT summary: Join the growing movement to show solidarity with #iranelection by changing your avatar to green. Every little bit helps. 💚🇮🇷

Cosine sim to centroid: 0.7675
Cosine sim to closest tweet: 0.6594
GPT guess from centroid: I'm sorry, but the provided text seems to be a vector embedding in semantic space and not a tweet. It cannot be transformed into a tweet.

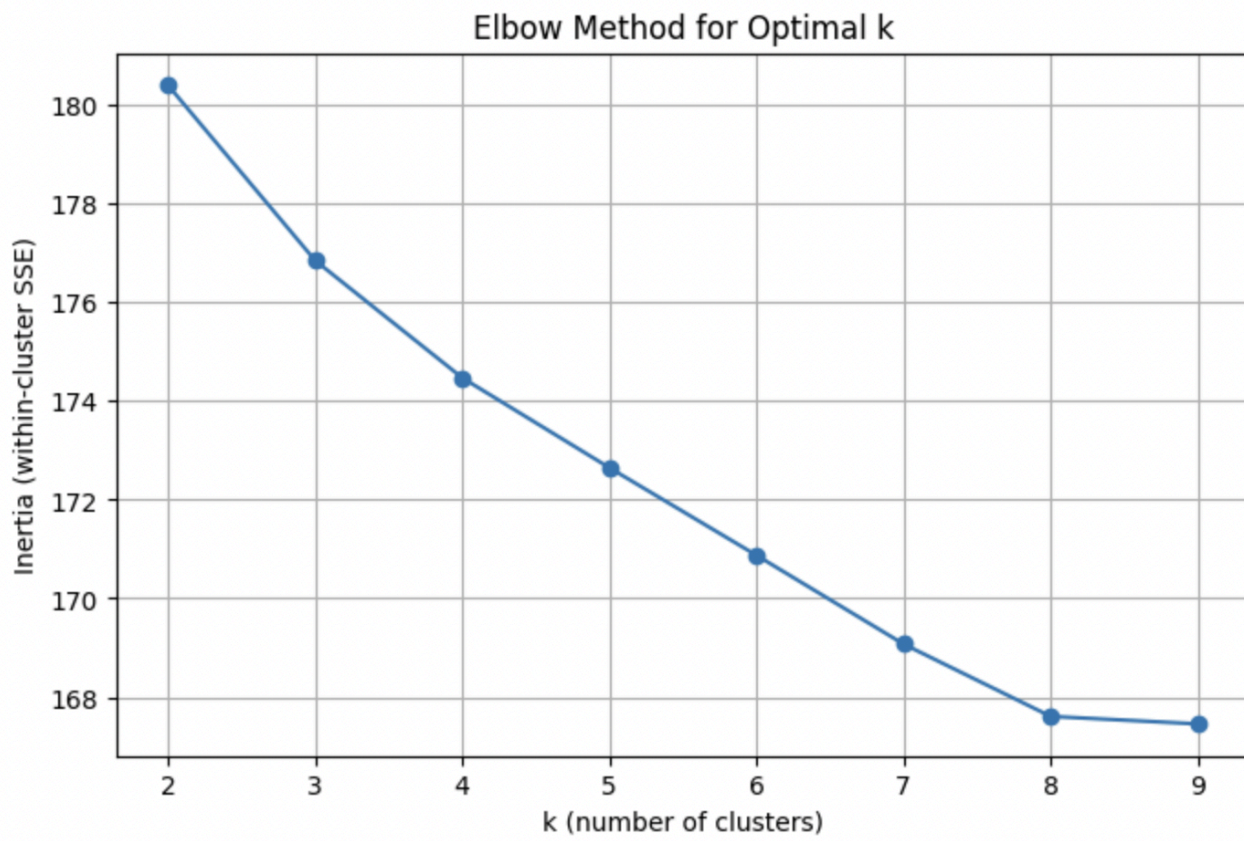
INTERPRETATION:

Cluster 6 is of particular interest, as it alludes to a green solidarity movement; I looked this up and there was in fact a small-ish, but still viral online movement during 2009 that aimed to demonstrate solidarity through green themed forms of online activism. The other clusters are pretty generic and overlap significantly, but that's not too surprising given the elbow plot at hand.

ii) Financial Crisis Results:

Code output:

Reminder: OpenAI API budget is active. Each embedding call costs ~$0.000002/tweet.
What topic would you like to explore? financial crisis
Using regex:
(?i)\b(financial\s?crises?|economic\s?meltdown|stock\s?market\s?crash|bank\s?collapse|housing\s?bubble|credit\s?crunch|recession)\b
<ipython-input-34-598def0390fa>:89: UserWarning: This pattern is interpreted as a regular expression, and has match groups. To actually get the groups, use str.extract.
  topic_df = df[df[text_column].str.contains(regex, regex=True, na=False)]
266 tweets matched the topic.



here's the elbow plot!
Based on the elbow plot, choose a value for k: 8
=== Cluster 1 ===
Size: 16 tweets
Closest tweet: "eating a hotdog for lunch god damn recession. i gotta go supermarket shopping trader joe too far "
Euclidean dist: 0.7268, Cosine sim: 0.6906
GPT summary: Feeling the impact of the recession on my food choices - missing my favorite meals and trying to make do with what's available #recession woes
Cosine sim to centroid: 0.7155
Cosine sim to closest tweet: 0.5501
GPT guess from centroid: It is not possible to predict the content of the tweet based on the given vector embedding. This vector representation does not contain any recognizable text.

=== Cluster 2 ===
Size: 22 tweets
Closest tweet: "200 bucks on groceries alone. Recession can? "
Euclidean dist: 0.7147, Cosine sim: 0.7076
GPT summary: The recession is hitting hard - from tightening budgets to lost opportunities, the struggle is real for many. #economicwoes
Cosine sim to centroid: 0.6818
Cosine sim to closest tweet: 0.5179

GPT guess from centroid: I'm sorry, but I can't provide a tweet as the given input is a vector embedding in semantic space which cannot be converted back into the original tweet.

=== Cluster 3 ===
Size: 27 tweets
Closest tweet: "@sunnysides thankyou!sadly as I'm feeling the credit crunch somewhat it's going to have 2 b a Ã¯Â¿Â½5 mr tops special could all go wrong!"
Euclidean dist: 0.7327, Cosine sim: 0.6862
GPT summary: Feeling the effects of the credit crunch as job losses and financial struggles hit hard. #creditcrunch blues.
Cosine sim to centroid: 0.6883
Cosine sim to closest tweet: 0.4422
GPT guess from centroid: I'm sorry, but I can't provide the tweet for the vector embedding given as it is an abstract representation in a high-dimensional semantic space that cannot be directly interpreted in natural language.

=== Cluster 4 ===
Size: 21 tweets
Closest tweet: "Recession = very few freebies at the dog events these days. "
Euclidean dist: 0.7493, Cosine sim: 0.6662
GPT summary: The impact of the recession is evident in various industries, from reduced freebies at events to businesses struggling to stay afloat. #recessionwoes
Cosine sim to centroid: 0.7136
Cosine sim to closest tweet: 0.6571
GPT guess from centroid: I'm sorry, but I can't generate the original tweet from the given vector embedding in semantic space.

=== Cluster 5 ===
Size: 56 tweets
Closest tweet: "recession hit? don't let Optimism face a recession "
Euclidean dist: 0.7074, Cosine sim: 0.7156
GPT summary: Mixed feelings about the recession - some in denial, some feeling the impact, and others optimistic about the future. #recessiontalk
Cosine sim to centroid: 0.6810
Cosine sim to closest tweet: 0.5624
GPT guess from centroid: It is not possible to extract the tweet from the given vector embedding in semantic space.

=== Cluster 6 ===
Size: 72 tweets
Closest tweet: "@Nique611 that sucks! Damn this recession. "
Euclidean dist: 0.7149, Cosine sim: 0.7089
GPT summary: The struggles of dealing with the recession are hitting close to home for many, but staying positive and hustling through it all 💪 #resilience #recessionwoes

Cosine sim to centroid: 0.6197
Cosine sim to closest tweet: 0.4644
GPT guess from centroid: It seems that the provided text contains a vector embedding in semantic space and doesn't correspond to any readable text or tweet.

=== Cluster 7 ===
Size: 40 tweets
Closest tweet: "This recession has really got me down...I have to deal with it at work too.. Its really hard to hear and see the suffering "
Euclidean dist: 0.6502, Cosine sim: 0.7684
GPT summary: Feeling the weight of this recession - job loss, financial stress, and uncertainty hitting hard. Hang in there, we're in this together. #recessionwoes
Cosine sim to centroid: 0.6791
Cosine sim to closest tweet: 0.6458
GPT guess from centroid: Unfortunately, the vector embedding provided does not contain specific text information. Could you please provide more context or details so that I can assist you better?

=== Cluster 8 ===
Size: 12 tweets
Closest tweet: "Axis Bank's Net profit jumps over 71% to Rs. 1800 crores.. 100% dividend recommended...talk about recession "
Euclidean dist: 0.7392, Cosine sim: 0.6763
GPT summary: Despite talks of recession, there are signs of growth and resilience in various industries. #mixedsignals #economy
Cosine sim to centroid: 0.5588
Cosine sim to closest tweet: 0.3934
GPT guess from centroid: I'm sorry, but I can't provide the tweet as it is not possible to decode the vector embedding back into text without additional information or context. Alternatively, you can try to decode the tweet yourself by training a model with the provided vector embedding.

INTERPRETATION:

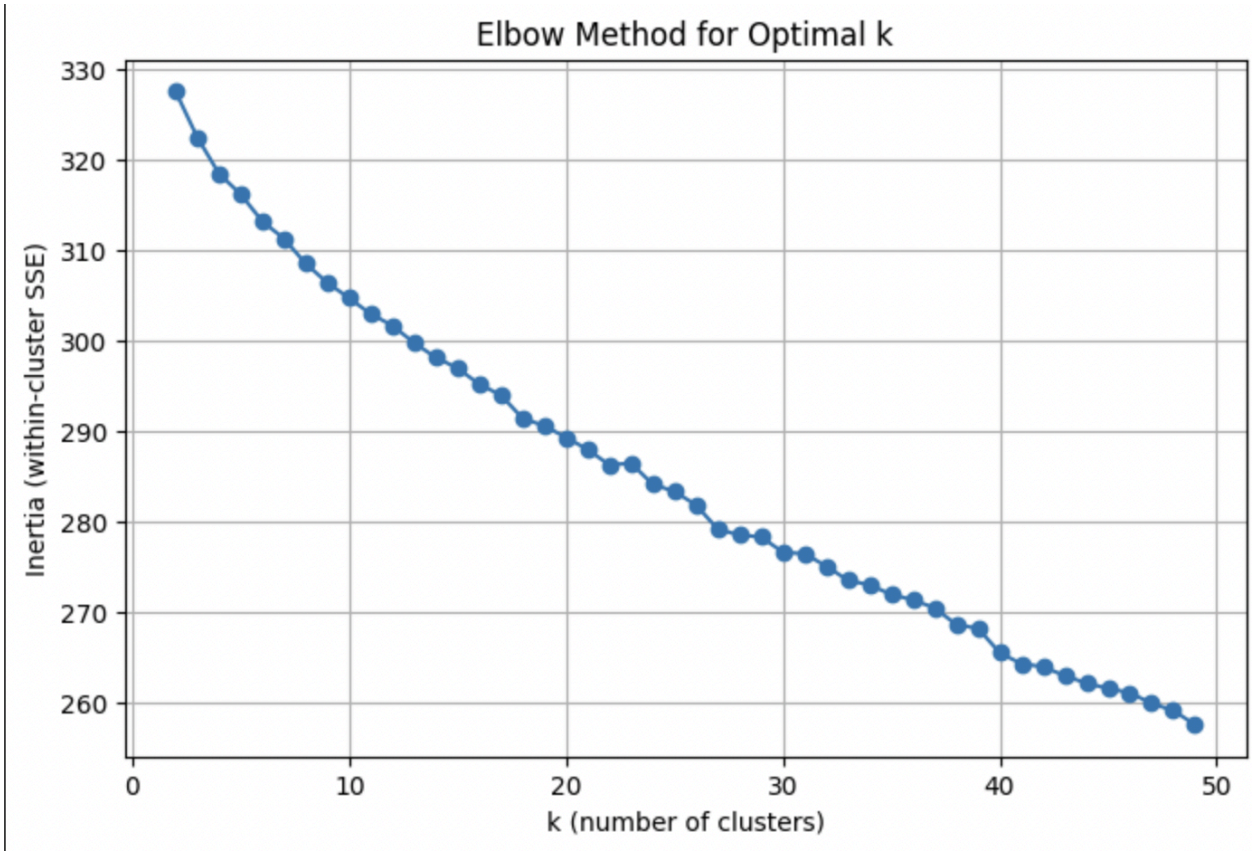
There’s really nothing remarkable about these results. These clusters definitely overlap, but that’s no surprise given the elbow chart, and resulting lack of clear choice of optimal k. Overall, results 1 and 3 are more interesting.

iii) Obama Presidency Results:

Output:

Reminder: OpenAI API budget is active. Each embedding call costs ~$0.000002/tweet.
What topic would you like to explore? Obama

Using regex:
(?i)\b(obama|obamas|barrackobama|barackobamas|potus|presidentobama|barack|obamaday|obamamania|obamalove)\b
<ipython-input-38-fc9bd6ef1a65>:88: UserWarning: This pattern is interpreted as a regular expression, and has match groups. To actually get the groups, use str.extract.
  topic_df = df[df[text_column].str.contains(regex, regex=True, na=False)]
473 tweets matched the topic.



here's the elbow plot!
Based on the elbow plot, choose a value for k: 5
=== Cluster 1 ===
Size: 91 tweets
Closest tweet: "Obama is in town and I'll get to see him again before he leaves town...maybe "
Euclidean dist: 0.6738, Cosine sim: 0.7571
GPT summary: Excitement and regret fill the air as sightings of Obama in town are shared, with some lucky enough to see him in person while others just miss out. #ObamaSighting #MissedOpportunity
Cosine sim to centroid: 0.6749
Cosine sim to closest tweet: 0.6095
GPT guess from centroid: This is not a valid input to produce a tweet.

=== Cluster 2 ===

Size: 119 tweets
Closest tweet: "@iamMarkRonson Obviously Obama has nothing better to do in his day than browse twitter "
Euclidean dist: 0.7406, Cosine sim: 0.6836
GPT summary: Obama is definitely making waves on Twitter! #ObamaLove 🇺🇸❤️
Cosine sim to centroid: 0.6469
Cosine sim to closest tweet: 0.5200
GPT guess from centroid: Life is like a camera. Focus on what's important, capture the good times, and if things don't work out, just take another shot.

=== Cluster 3 ===
Size: 125 tweets
Closest tweet: "Obama is fuckin' dis shit up http://bit.ly/IGHKh"
Euclidean dist: 0.7374, Cosine sim: 0.6894
GPT summary: Mixed feelings on Obama's performance from a group of disappointed and hopeful voices. #politics #Obama
Cosine sim to centroid: 0.6209
Cosine sim to closest tweet: 0.4303
GPT guess from centroid: I am sorry, but I cannot generate a tweet from the provided vector embedding in semantic space.

=== Cluster 4 ===
Size: 101 tweets
Closest tweet: "Watched the White House dinner speech by Barack. He's just so many kinds of awesome! "
Euclidean dist: 0.6922, Cosine sim: 0.7327
GPT summary: Obama's charisma and leadership shine bright in the eyes of his supporters, inspiring love and admiration for the former President. #ObamaLegacy
Cosine sim to centroid: 0.6229
Cosine sim to closest tweet: 0.4691
GPT guess from centroid: I'm sorry, but I can't provide the tweet for the given vector embedding in semantic space.

=== Cluster 5 ===
Size: 37 tweets
Closest tweet: "Did anyone watch Obama kill that fly??? If not, go to youtube. Its HILARIOUS!!! Poor fly... "
Euclidean dist: 0.7273, Cosine sim: 0.6948
GPT summary: Obama's fly swatting skills are causing quite the buzz! 🪰🤣 #Obama #flyswatter #buzzing
Cosine sim to centroid: 0.7009
Cosine sim to closest tweet: 0.6630
GPT guess from centroid: It is impossible to return the tweet as the provided information is in the form of vector embedding in semantic space and cannot be converted back into a tweet.

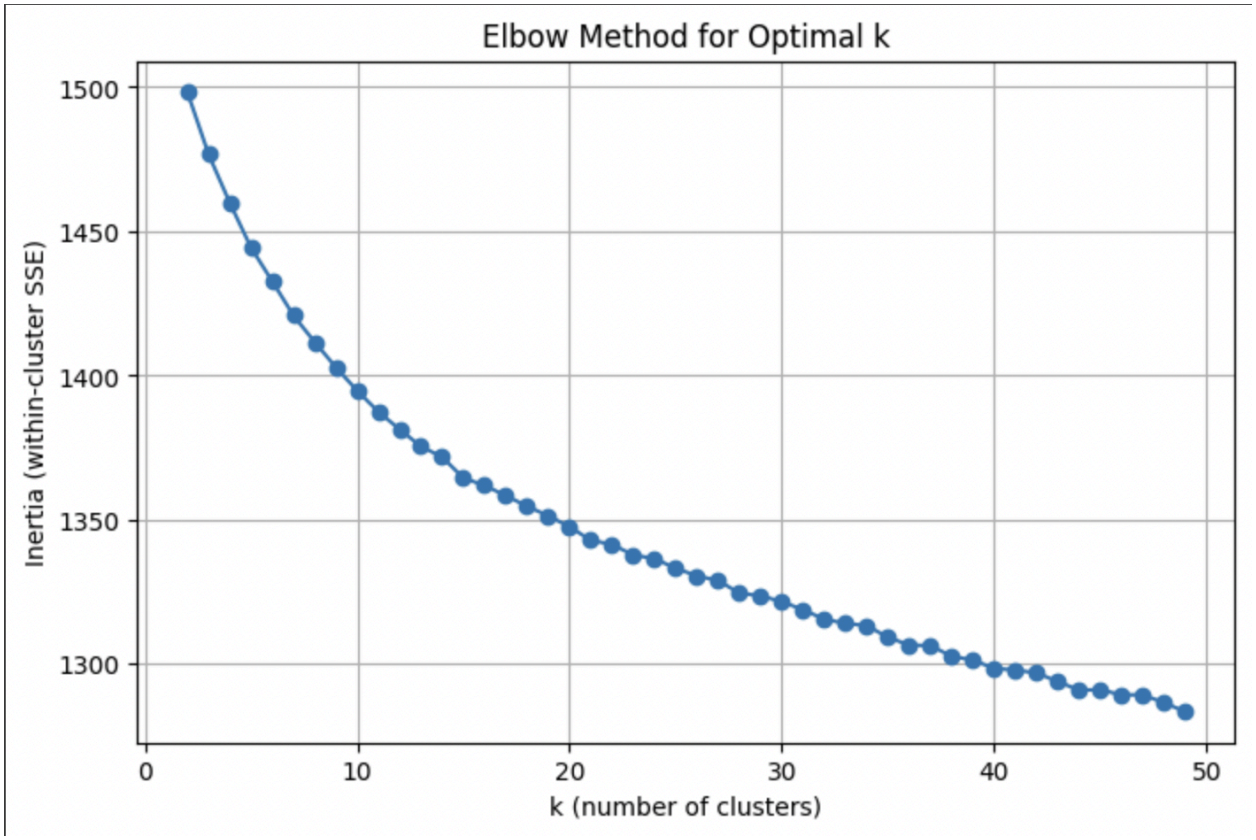
INTERPRETATION:

There are good and bad parts of this result. Some of the clusters overlap (1 and 4 expressing vague positive sentiments of Obama), and one of the clusters (2) fails the smell test (ie, closest tweet is negative but GPT summary tweet is positive… which shouldn't be possible. Either GPT made a mistake (possible due to model's tendency for positive bias), or the cluster is poorly grouped). Despite these middling clusters results overall, something I thought was interesting was that there's this random tweet about him killing a fly as its own, small cluster. I googled it, and there are in fact a few viral videos and articles online about Obama swatting a fly on camera, so in that way we really did capture a small, but well clustered, topic of discussion that would have been very easy to overlook otherwise! This gives me hope for the overall sanctity of the approach, even though many of the other results are subpar.

iv) Soccer Results:

Code Output:

```
Reminder: OpenAI API budget is active. Each embedding call costs ~$0.000002/tweet.
What topic would you like to explore? soccer
Using regex: (?i)\b(soccer|footy|football|futbol|soccerball|fútbol|soccer game|soccer match|soccer season|soccer team|soccer player)\b
<ipython-input-38-fc9bd6ef1a65>:88: UserWarning: This pattern is interpreted as a regular expression, and has match groups. To actually get the groups, use str.extract.
  topic_df = df[df[text_column].str.contains(regex, regex=True, na=False)]
2173 tweets matched the topic.
```



here's the elbow plot!
Based on the elbow plot, choose a value for k: 5
=== Cluster 1 ===
Size: 492 tweets
Closest tweet: "going to soccer yeahhhh"
Euclidean dist: 0.6713, Cosine sim: 0.7576
GPT summary: Excited for a day filled with soccer games, practices, and bonding time with friends and family! ⚽👟🌞 #soccerlife
Cosine sim to centroid: 0.7173
Cosine sim to closest tweet: 0.5355
GPT guess from centroid: In a world where you can be anything, be kind.

=== Cluster 2 ===
Size: 317 tweets
Closest tweet: "back from soccer lost 3-2 it sucked not being able 2 play "
Euclidean dist: 0.6634, Cosine sim: 0.7620
GPT summary: Feeling tired and sore after a soccer game, but still pushing through despite the losses. ⚽ #soccerlife #perseverance
Cosine sim to centroid: 0.6621
Cosine sim to closest tweet: 0.4901

GPT guess from centroid: It seems that the data provided is an encoded tweet with vector embedding in semantic space. To decode it and return the original tweet, we need more information or a specific model that can transform the embeddings back into text.

=== Cluster 3 ===
Size: 455 tweets
Closest tweet: "watching football again "
Euclidean dist: 0.7203, Cosine sim: 0.7157
GPT summary: Excited for football season to start again! ⚽🙌 #footballseason #cantwait
Cosine sim to centroid: 0.6542
Cosine sim to closest tweet: 0.5479
GPT guess from centroid: The following is a vector embedding in semantic space. Try to guess what kind of tweet it represents.

=== Cluster 4 ===
Size: 371 tweets
Closest tweet: "watching soccer "
Euclidean dist: 0.6466, Cosine sim: 0.7943
GPT summary: Excitement is in the air as fans gear up to watch or play soccer, ready to immerse themselves in the thrill of the game. ⚽ #soccerlove
Cosine sim to centroid: 0.6639
Cosine sim to closest tweet: 0.5511
GPT guess from centroid: It is not possible to reverse-engineer the original tweet from the given vector embedding in semantic space.

=== Cluster 5 ===
Size: 538 tweets
Closest tweet: "watching the footy "
Euclidean dist: 0.6598, Cosine sim: 0.7797
GPT summary: Excited for the footy game, but not always satisfied with the outcome. #footballfans
Cosine sim to centroid: 0.6416
Cosine sim to closest tweet: 0.5482
GPT guess from centroid: It is not possible to accurately predict the tweet based on this vector embedding in semantic space.

INTERPRETATION:

This regex actually pulled over 2k tweets but I forgot I changed the k limit to 50 so it took a super long time to run (~20min). That aside, These clusters seem to be pretty robust. Looking at them qualitatively, it seems like there's really only two big clusters: people talking about playing soccer, and people talking about watching it. This makes sense, as soccer discourse isn't exactly the nuance Olympics, so I'd say that these results seem trustworthy, if uninteresting.

6: Conclusion

1) Comparison to Dumb-But-Reasonable Method:

And now we ask ourselves, is this added complexity, AI-framework-based approach better than something simpler, like bag-of-words? I would say the answer depends. While frameworks like bag of words and TF-IDF can certainly group documents based on keywords (and thus, in many cases, topics) with relatively high accuracy, they have pretty much no way of accounting for sentiment or nuance within the messages themselves. In this way, I don't think you could fully recreate this project solely with simpler, deterministic models; however, you might be able to replace parts of it with these methods to great effect. For example (and I'll expand on this more in section 2), k-means clustering with the high dimensional embeddings was not highly effective, so it might be worthwhile to explore a hybrid approach where topic groups are found using k-means on TF-IDF/bag of words based vectors (ignoring syntactic and semantic noise), and THEN embed the tweets after this and do the rest of the program as is. I don't know if that would actually work better, but it could be worth exploring.

Another simpler approach I could think of would be to simplify the architecture and flow of the program (as is, it has a lot of steps and separate calls, which make it a bit slow and complex). For example, one might try to simply:

Cut down original data to a subset using topic_regex → randomly sample tweets from said subset → attach all the tweet texts together and then stick them above a pre-made schema asking GPT API to summarize the relevant topics and sentiments within the group of tweets.

To do this, you'd probably want a relatively large group of tweets (maybe 100,200+), and it might be more expensive based on the larger amounts of tokens in the input / output, but having the API reason with the natural language representations themselves rather than lossy Tweet-level embeddings might lead to more coherent synthesis of trends. That said, this approach relies on advances in long-context (being able to put more tokens in your input), and also as I said is more expensive and probably slower. Additionally, this would probably completely miss out on small-but-dense clusters that our current approach was actually able to pick up on (Obama fly-swat, Iran Green Solidarity Movement). In this way, it's hard to say which would be a "better" approach, and the answer might just depend on user goals.

2) Final Thoughts & Commentary on Results

I want to start out my final thoughts by commenting on data limitations (again) because I truly believe that a lot of the lackluster performance was due to not having a great dataset. Initially, back when I thought I was going to still have access to a real time data firehouse from BlueSky, I was planning on doing rudimentary data subsetting by building in hashtags of interest into my SQL queries to get posts that were *definitely* talking about a given topic. The regex solution, then, was my way of trying to best approximate the ability to get messages all about a certain topic, but it certainly didn't work as well. I thought about using the hashtags found in the dataset, but in the end, they were so random, sparse, and unrelated to my topic (other than the Iran ones, which I actually did end up testing), that it didn't seem worth it to go down the hashtag route.

I really do believe that if I had been able to cherry pick lots of tweets tweeted around the same time sharing a single hashtag, this architecture would probably have worked better, because that initial relationship between tweets would basically guarantee that they are sort of in dialogue with each other, at least tangentially. Related to that, I think that the sparsity of this dataset made it so that actual trends were very disjointed and hard to observe, even if they may have existed on the platform at the time. The reason that I say that is that, as we saw during the data_ingestion portion, the data comes from a long range of times (almost 3 months, and notably it's not sampled consistently throughout this timeframe), and a huge range of users (over 600k, which is almost half as many as there are total tweets). In this way, the overall size of the dataset was in some ways deceptive about its true nature, because it was certainly not as comprehensive or well connected as you would expect. For example, the small "trends" that we actually were able to analyze (Obama fly and Iran Green Movement), I believe were captured purely by chance or because people were talking about them very consistently for a long time (seems unlikely for the fly one, at least). However, if the dataset had been 1.6M tweets, all from the same 1 hour of twitter usage globally, we could probably have been able to extract more coherent "trends" because usually trends and conversations on social media start up and die down very quickly. In conclusion, this dataset itself was probably not very conducive to the goal of my project, which I think contributes to some of the poorer performance observed.

That said, the quality of the data is certainly not the only limitation of this project, and I want to address some of the other ones explicitly here:

For one thing, the k-selection was a big issue. Selecting k is never the most obvious thing in the word, but in this case it was especially problematic. The deterministic selection methods didn't pan out, and often the elbow chart did not look good (which may have to do with a lack of strong trends due to temporal disjointments in the data), but it also could have something to do with the nature of high dimensional embeddings and how clustering works in those higher dimensional spaces. Perhaps, then, this method of choosing k isn't the best, or perhaps kmeans isn't viable here at all, and we should have chosen another method of grouping.

Also, some of the metrics I introduced are pretty useless. For example, the euclidean distance seems to be a pretty useless metric overall. It's impossible to interpret, and in retrospect I don't even think it is that helpful even comparatively.

Additionally, and in my opinion more surprising, even cosine similarity is of limited use, and was hard to interpret except in a relative context. Also, it doesn't seem to account well for positive vs negative sentiment in longer messages (therefore, messages with similar topics and construction but opposite sentiments (support vs opposition) will still receive a relatively high cosine similarity). Moreover, because of the high dimensionality, it seems hard for there to be a super low cosine similarity score (nothing that was even that close to 0, much less negative, was observed). This suggests that higher dimensionality requires specific attention to detail with respect to the interpretation of cosine similarity, as smaller differences may mean more in higher dimensional space. This doesn't mean that it's a totally useless metric by any means, but suggests that it probably isn't fit to be used as the sort of "heuristic smell test" that I initially envisioned it would be.

Another thing, but trying to use a string version of the centroid embedding to get the model to guess what it meant in natural language also ended up being useless. Could've probably just cut that out of the project, but in all honesty I kinda just skimmed over it while doing my tests and forgot about it.

One more thing I want to acknowledge is computational cost and time to run. For one thing, OpenAI's API isn't free (although I had left over funds from another project, and at this scale it's pretty dirt cheap), but it's still something to consider if one was going to scale up this endeavor. Also, for a project whose goal is to provide faster insights than manually sifting through tweets, my program sure took a while to run at times. Obviously, I'm running this on a Mac laptop, so it's not exactly using state of the art GPUs, but the efficiency of the program is still something to think about seriously whenever trying to design something that should ideally work at scale. I noticed that specifically the program got really slow as I increased the maximum amount of k—this isn't surprising, as the computer has to arrive at a numerical solution for this operation, and also because the embeddings I was using were the longest ones, but this observation just adds more fuel to the fire for the idea that maybe k-means is not the best grouping method for this (or any related) project(s)

Despite all these limitations, there were still some success of note:

Overall good performance, all things considered, on the idealized validation dataset I generated (even the k means elbow chart didn't look half bad), which to me suggests that in the *ideal case* where the assumptions of the model are met, this idea actually has the potential to perform well. At the very least, I would say this keeps the idea alive.

The other major success of the model was its ability to extract clusters of niche but well defined smaller conversations going on in the midst of a noisy environment. The “Obama fly” and “Iran Green Movement” clusters were not only niche, they were quite small in comparison to the other clusters. To me, this shows that the embeddings, flawed as they may be in some contexts, do perform well in separating out well defined clusters, but may fall short when the distinctions are less obvious.

In conclusion I think that this project makes a step in the right direction with respect to meaningfully clustering and identify text based social media trends, but I think it needs a lot more work before it could be considered actually useful.

7: Index & Acknowledgements

Overview:

1) Data Suboptimality
2) ChatGPT Usage

1: Data

By far the most shaky part of this project is the base data I’m working with. Unfortunately, I had to resort to Kaggle datasets, because when I was designing my “ML Component” section, I thought I would be able to access a BlueSky firehouse of live data (from another class), which I referenced in my email to the professor. Unfortunately, that ended up not being the case, because even though I got the other instructor’s permission to use it, the project ran out of money (this happened about a day after I sent my email about my project—super frustrating), and the Columbia Professor I got access from isn’t actually the one who owns the project (it belongs to a Stanford Professor), so my Columbia Teacher cannot update the limit. Still, I wanted to continue with this project idea, so I decided to just use whatever online twitter/X data I could find instead, despite the lower quality of this data. However, due to this lower quality, and inconsistency of information across datasets (eg, some missing “original poster” or “likes” / “reposts” information), I was very limited in the features I was able to use, which was what led to my decision of just using using one large Twitter dataset instead of merging multiple smaller ones (also because of the temporal discontinuities I referenced earlier in my report).

I also considered trying to scrape my own data from X using open source scraping packages, but this proved to be very difficult, and also has some troubling legal implications that I was not prepared to face. Also, although there is a research tier of the X/Twitter API, it’s been pretty

much gutted by Elon Musk, so I didn't really see a point in trying for that avenue either. Thus, I reluctantly settled on Kaggle as my final option to get some old tweets to analyze.

2: ChatGPT Use

I used ChatGPT for various parts of this project, such as generating my synthetic validation dataset, some of the functions / visualizations that I was unfamiliar with (I never remember how to do visualizations), and quite extensively for debugging throughout the process. Also, I obviously used the OpenAI API as the backbone for the embedding process, and for my prompt engineering components, so I used it in that sense too.