12/16/24

EMT-678 Final Project: Historical Sentiment Analysis

Wyatt Blair

Introduction

Goal

Every opinion on Earth exists in the context of those which came before it. This project aims to create a time efficient pipeline which can be extended to understand how historical sentiment has changed over the years in the English speaking world. In order to discern any meaningful pattern, a large volume of text spanning a number of decades needs to be analyzed. Due to the need for a large dataset, a secondary goal of making the pipeline as scalable as possible emerges. In the end, the pipeline will produce a line chart, demonstrating how positive or negative the sentiment of each year was as it pertains to a pre-selected topic. Additionally, the pipeline will be run on increasingly larger subsets of the dataset, with each step of the pipeline being timed in order to measure how well the pipeline holds up under scale. The repartition size of the initial DataFrame will also be varied through these time-trials to understand which partition size works best for a given dataset size.

Data Source

The Project Gutenberg dataset contains 70,000 free plain-text books online which constitutes the bulk of the dataset used in this project. In order to build the dataset, I constructed a Python script which scrapes the Project Gutenberg website. During the scrape, the plain text of the book, as well as the year of publication, and some other useful information is organized into a Pandas DataFrame, which is then saved in a .csv file. The pipeline is only concerned with the "year" and the "text" columns of this .csv so if a user wants to add additional books in the future, all they would need to do is add the year of publication and the text of the book to this table-- the source does not even need to be from Project Gutenberg. The implication being that

content such as modern day tweets, forum posts from the 90s and the 00s, or any other form of English text with a date attached to it could potentially be added and analyzed. I kept only English texts and limited the dataset to 15,000 books in total. This .csv takes up about 3.8 GB, with each full book text contributing the most to the size. Due to the size, the .csv is split into five equal-size (or as close to equal as possible) .csv which can be uploaded to DataBricks independently.

In order to achieve a time-efficient run of the pipeline, there are no models which are trained during execution of the pipeline. Instead, lemmatization and sentiment analysis are performed with the use of two open-source datasets containing common lemmatizations and a sentiment ranking map. For lemmatization, I used the Universal Dependencies English Web Treebank, which was compiled by The Leland Stanford Junior University. The UDW corpus comprises 254,820 words and 16,622 sentences, taken from five genres of web media: weblogs, newsgroups, emails, reviews, and Yahoo! Answers. In order to assign a sentiment score to each text for a given topic, I used the AFINN-111 dataset. This dataset contains about 3,000 words with an assigned “sentiment score” (ranging between -5 and +5) which can be used to assess the sentiment of a given body of text. These mappings, while crude, allow for very quick computation of sentiment across all of the text in the .csv and could conceivably be extended in the future with more lemma and sentiment mappings if these prove insufficient.

Pipeline Architecture

The pipeline is broken into a few different phases. Once Spark is started, the pipeline first loads all the partial .csv files and recombines them into one large DataFrame. All columns except for the “year” and “text” column are dropped to keep the DataFrame as light-weight as possible. Some standard data-cleaning is performed on the “text” column such as removing the

title page which every Project Gutenberg entry has, removing any strange symbols, and other standard data-cleaning practices. From there, the “text” column is tokenized and exploded so that every row is a token and an associated year.

Now, the DataFrame is in an appropriate state to use the lemmatization and sentiment analysis process. To begin, the lemma mappings are loaded and stored in a DataFrame where one column is the lemma and the other column represents a list of words which should be mapped to the corresponding lemma. This lemma-DataFrame is used in a join to replace all tokens with their corresponding lemmas. Doing so increases the likelihood that the sentiment analysis pipeline will appropriately flag and score tokens appropriately. In order to perform this computation, the sentiment-analysis frame is created which contains lemmas in one column and a sentiment score between -5 and +5 in the other column. However, before the sentiment can be applied to each topic, the topics themselves must be defined. This is accomplished through a third DataFrame, called the topics-frame. The topics-frame has one column with the name of the topic and another column with a plethora of words related to that topic. The pipeline searches through the lemmas to find one of these topic-related words, aggregates 15 tokens before and 15 tokens after the topic-related word, uses the sentiment analysis frame to score the whole “phrase” and then adds that score to the year’s running total. In this way, a sentiment score is assigned to a given year for a given topic and is then averaged by the number of occurrences of the topic in that given year so that it is comparable to other years’ scores. Finally, all of these sentiment scores are plotted in a line chart showing how the sentiment changes throughout the years.

Each step of this process is measured for time-efficiency under scale by wrapping each step in a Python “stopwatch” (a module called timeit). Then the repartition size and the size limit of the original DataFrame is varied so that these values and their impact on the runtime of the

pipeline can be measured and analyzed. These results are presented in a heatmap which demonstrates how a specific combination of size limit and repartition size yields different runtime behaviors for different steps of the pipeline process. The pipeline is separated into the following steps: spark startup, data load, lemma load, sentiment load, topic load, compute sentiment, and plot.

This pipeline was designed to leverage Photon Acceleration and Arrow wherever possible in order to cut-down on runtime. In that way, it uses entirely native PySpark functions and avoids User-Defined Functions meaning the full optimization of PySpark can be unleashed.

One more important factor to mention is the uneven distribution of book titles across years. The majority of books in the Project Gutenberg dataset are newer (published past the year 2000). In order to combat this, a sampling strategy is implemented where the representation of years in the DataFrame are as even as possible.

Body

Sentiment Analysis

Several topics were selected for sentiment analysis: gender, politics, education, technology, religion, medicine, science, and philosophy. Each of these topics had several associated words mapped to it so that it could be discovered during the pipeline run. For example, the topic of religion had the following words mapped to it: "religion", "church", "god", "bible", "pray", "faith", "holy", "divine", "priest", "deity", "temple",... (and about 20 others). Unfortunately, of the 8 topics tested, only 5 produced meaningful results. This is likely due to the

lemma mappings and sentiment scores not containing words which map to these topics well and leaves room for improvement in that regard.

Of the topics which were analyzed, a few standout. Namely, the medicine topic tends to be overwhelmingly negative in terms of sentiment which I believe to be a consequence of the design of the pipeline. Since medical terms are often used near or in reference to illness and death, these tokens are likely being included in the 30-word aggregation around the topic word and are thus dragging the score down. In the opposite regime, the topic of “religion” tends to be overwhelmingly positive; likely for the same reason. Words like “faith” and “divine” are often used in sentences or phrases which contain other, positive sentiment words.

However, this project is not meant to simply discern whether a topic is or isn’t positive; it’s designed to look at how this trend changes throughout the years. The topic “politics” follows an interesting trajectory. It peaks in the late 70s / early 80s, and then plateaus from the mid-1990s to the early 2010s. Throughout the 2010s, the sentiment rises to a higher plateau where it stays until it crashes almost to zero in 2020.

Technology is another topic which follows an interesting trajectory in its sentiment. First of all, the pipeline was unable to locate terms related to technology prior to the early 1990s, likely due to limited terminology in the topic mappings. A significant peak was reached in the year 2001 which the technology sentiment never returned to. A local minimum in sentiment was attained in the early 2010s, out of which the sentiment has been steadily climbing ever since.

Philosophy appears to follow boom and bust cycles when it comes to sentiment, likely due to popular trends which are rooted in certain philosophies. Across the span of 30 years from

the early 1990s to the 2020s, there are about five of these boom and bust cycles almost perfectly spread out with about six years separating each boom.

To review these charts yourself, please refer to Section A in the Appendix.

Time Trial Analysis

In order to determine how this pipeline scales with time as more data is included– a step specific time trial function was created. This function takes two parameters: the repartition size and the dataset limit size. Using the given parameters, the full pipeline is run end-to-end with each step's runtime measured as it runs.

If one examines the charts in Section B, they will see that while there is a good deal of variety in the runtimes, one particular step contributes the most to runtime: plot time. This is certainly due to the fact that the only `.toPandas()` call in the entire pipeline occurs here meaning the final DataFrame must actually materialize and all the transformations on the intermediate DataFrames need to take place. Some steps have almost no variation in their runtime such as the start Spark step, which is to be expected. Given the tested sizes, a repartition size of 100 appears to be optimal for this particular dataset and seems to scale the best with dataset size. Further testing, which is not reflected in the charts in the Appendix, revealed an optimal repartition size of 200. The longest run of the pipeline (with a repartition size of 5000 and a dataset size of 10000) took a little over 7 minutes to complete. It is important to note here that the `.limit()` function itself takes time to run and so would also contribute to the results of scalability testing.

Future Work

There is a lot of room for improvement in this project. First of all, the topic flagging mechanism feels too simplistic since it is just using a simple hand-coded dictionary. If I could find a time-efficient and scalable embedding algorithm which embeds each token, I could attempt some form of clustering which looks for any tokens sufficiently “close” to the topic word so that it can pick up more results for sentiment analysis. Sentiment Analysis itself is also an area where the pipeline could be improved. No machine learning models are being used in this pipeline, only a simple mapping of sentiment score to word which was taken from an open-source repository online; but not for lack of trying. In experimenting with different sentiment analysis models, I found the inference time took far too long and did not scale well with larger datasets. The current methodology of loading a sentiment mapping proved far more time-efficient, albeit at the cost of performance. The same can be said for the lemmatization in the pipeline. If a more time-efficient and performance boosting version of lemmatization exists, it would be able to fit perfectly into the pipeline; however, I was unable to find one. Finally, the last (and perhaps the most obvious) improvement to the project would be to add more data. As mentioned in the Introduction, all that is needed to fit into the .csv is a year and a full body of text, which means not only could I extend this dataset with the other 60,000 books in Project Gutenberg, but I can add text from other sources like social media, textbooks, magazines etc. as long as they have an associated publishing year.

Conclusion

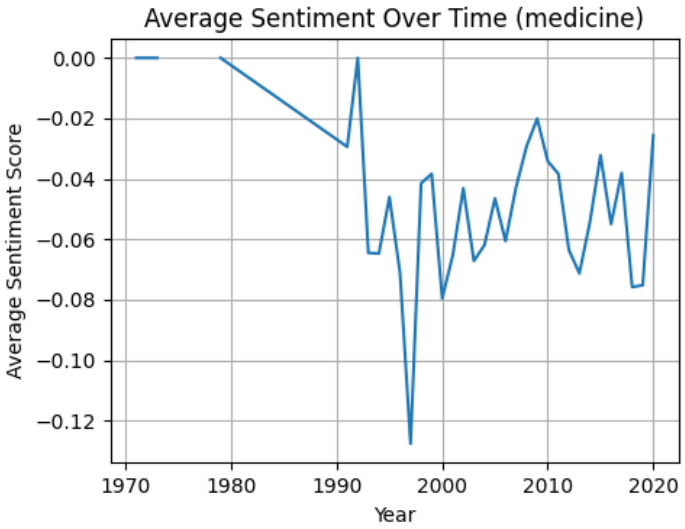
Opinions reinforce opinions, yet they exist to describe ever-shifting topics. These two forces shape what we call sentiment. Tradition and “common sense” exist because humans that

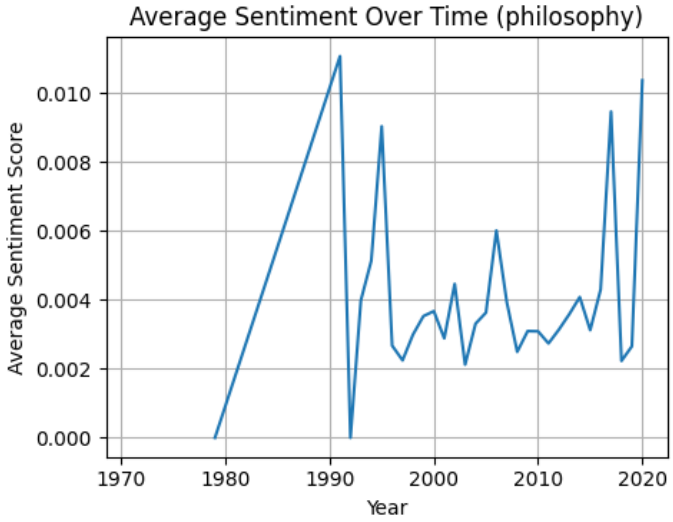
came before influenced their descendents' opinions with opinions they held to be true; but as the world changes, so too do these opinions. These fresh new ways of looking at and perceiving the world push back against traditional perceptions and opinions, challenging and eroding them. The middle ground that a group of people find themselves in is the public sentiment. The purpose of this project was not necessarily to perfectly measure the public sentiment of each topic, but rather to begin building a foundational pipeline through which a very large amount of data could be passed yielding a clearer and clearer picture of the public sentiment surrounding a topic. In the extreme, every piece of English text written from the 1970s (or some other start-date) through present day could be passed into a pipeline like this to get a very clean picture of public sentiment over the years. Such a dataset would be truly massive and so scalability is of the utmost importance.

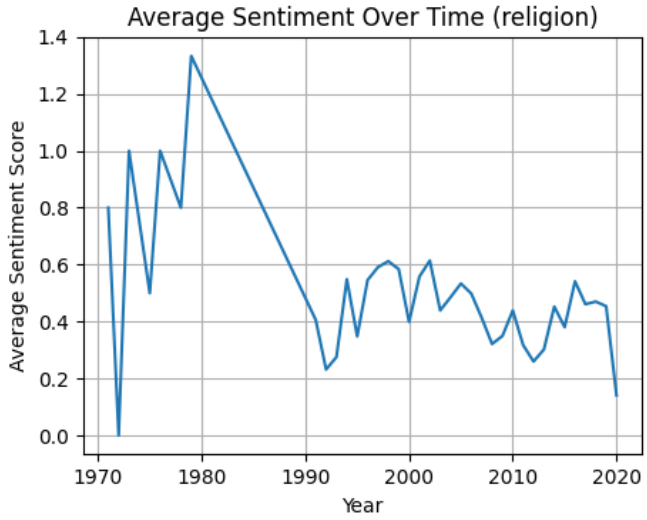
But just how well does this pipeline scale? For fun, I asked GPT-o1 to produce a Fermi Approximation for the number of bytes of English text written on Twitter since Launch and based on its chain of thought (which seems reasonable having checked the math myself) it produced the following number: 10^13 bytes (ten terabytes). For reference, this subset of the Project Gutenberg dataset used in this experiment was about 3.8 GB. Assuming a linear growth time in scale, the best performing pipeline configuration had a slope of ~300s/3.8GB meaning a runtime of ~25 million years for all of the English which is, obviously, not tenable. At least it's better than the worst performing pipeline configuration which had a slope of ~425s/3.8GB which would be about ~35 million years if every English tweet was run through it. In reality, these massive runtimes are the consequence of not yet taking full advantage of the parallelization available on a platform like DataBricks which would require paying for more computational power. Increasing the number of workers could reduce these runtimes by orders of magnitude, could invalidate the assumption of linear runtime, and ultimately result in reasonable runtimes.

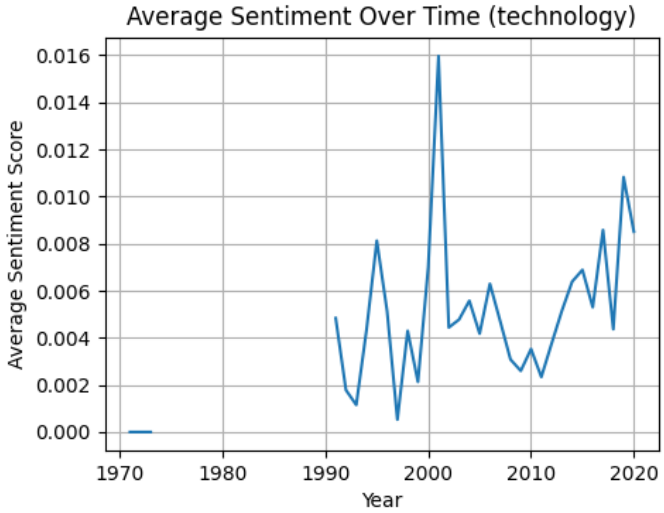
Appendix

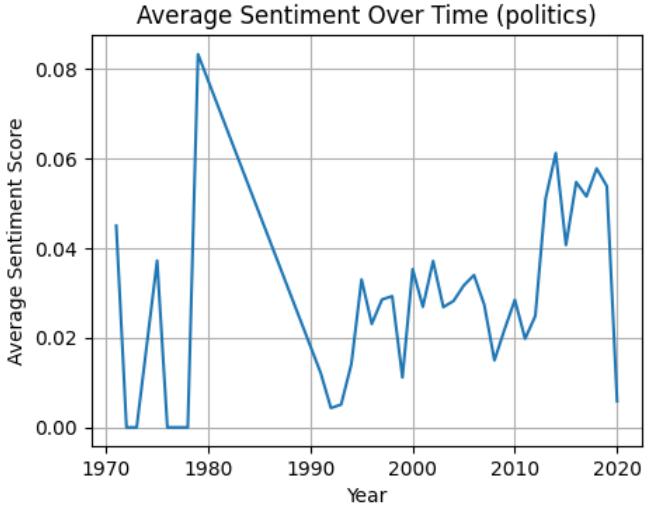
Section A (Sentiment Analysis)



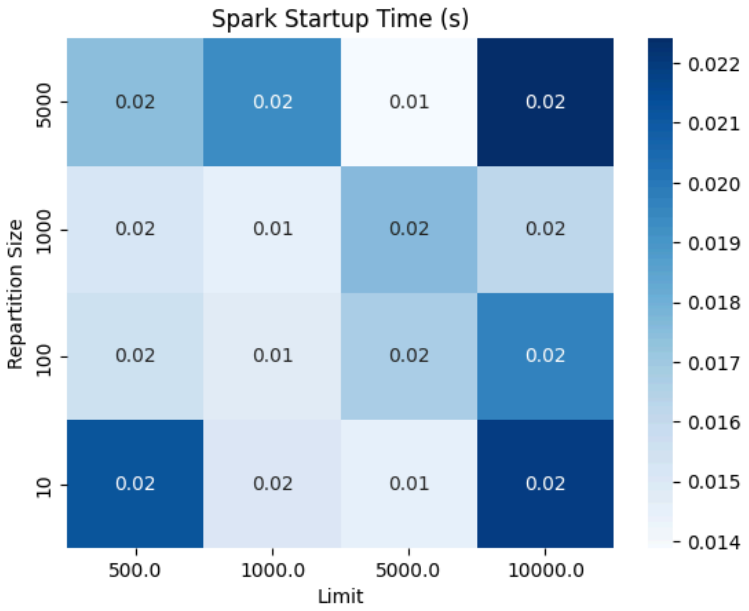


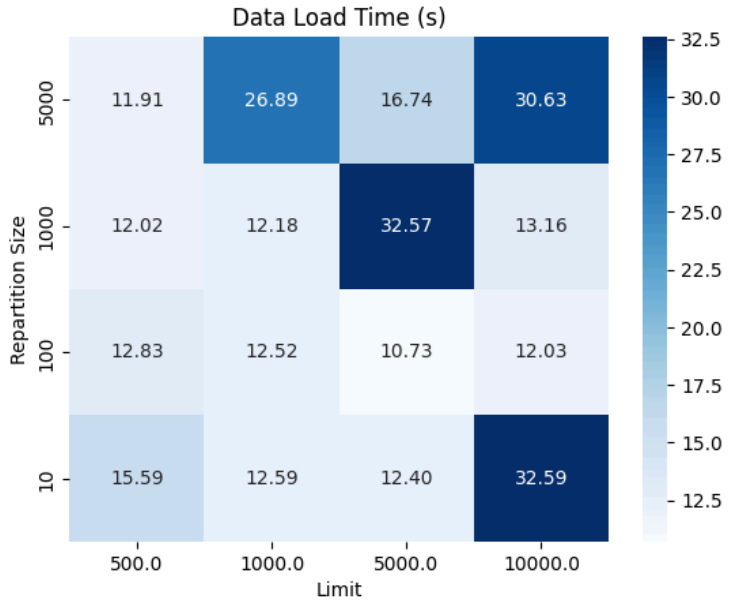

Average Sentiment Over Time (religion)
Average Sentiment Score
Year
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1970
1980
1990
2000
2010
2020

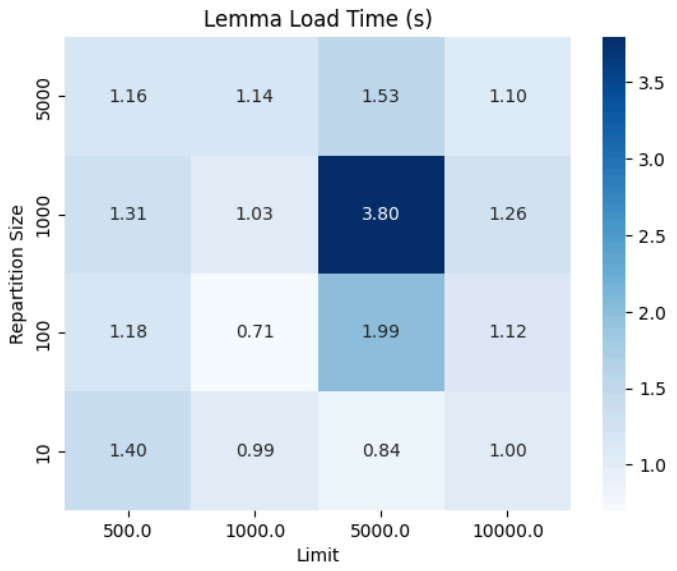

Average Sentiment Over Time (technology)
Average Sentiment Score
Year
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
1970
1980
1990
2000
2010
2020

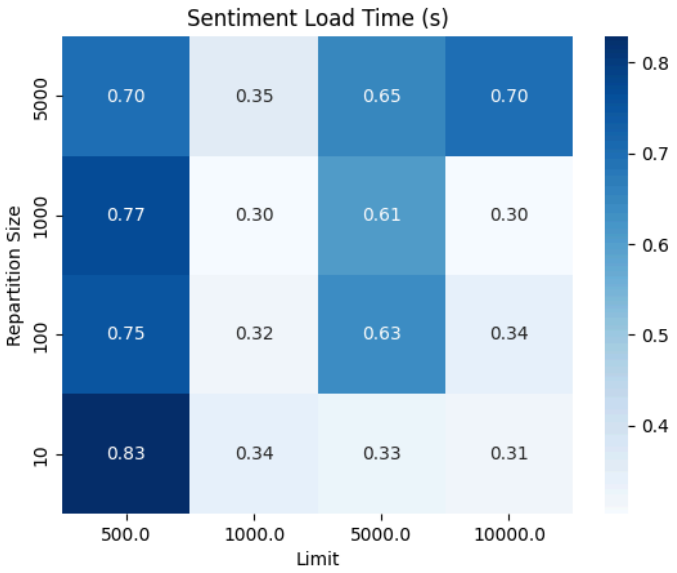

Average Sentiment Over Time (politics)
Average Sentiment Score
Year
0.08
0.06
0.04
0.02
0.00
1970
1980
1990
2000
2010
2020

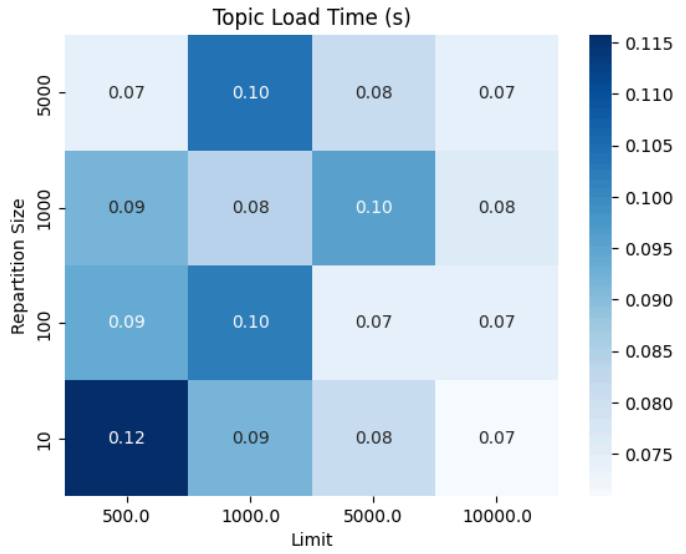
Section B (Runtime Analysis)

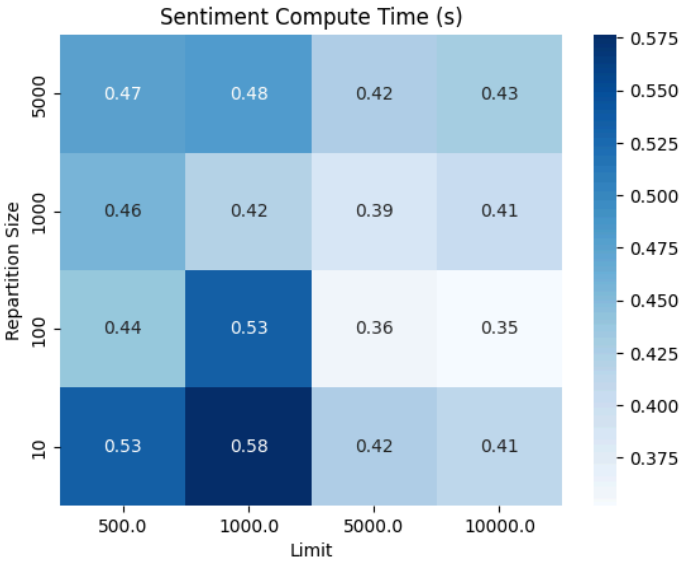


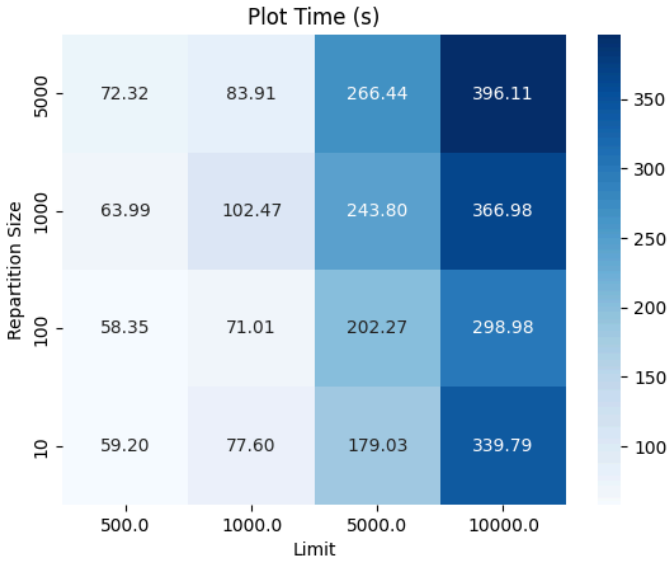


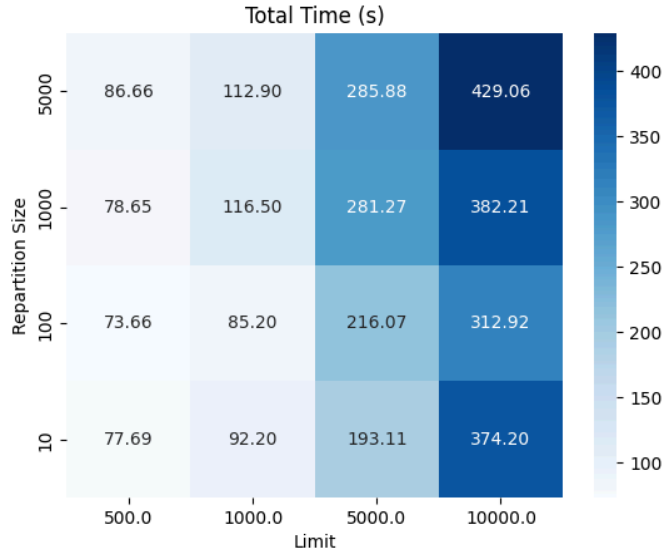

Lemma Load Time (s)
Repartition Size
5000
1000
100
10
1.16
1.14
1.53
1.10
1.31
1.03
3.80
1.26
1.18
0.71
1.99
1.12
1.40
0.99
0.84
1.00
500.0
1000.0
5000.0
10000.0
Limit
3.5
3.0
2.5
2.0
1.5
1.0

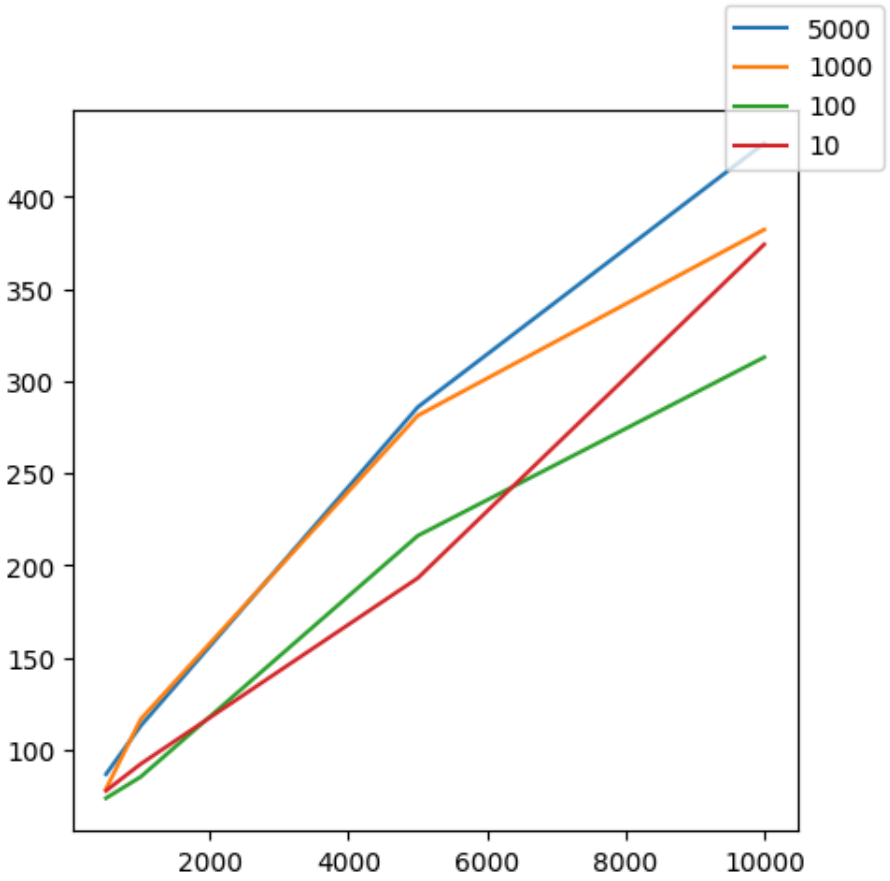

Sentiment Load Time (s)
Repartition Size
5000
1000
100
10
0.70
0.35
0.65
0.70
0.77
0.30
0.61
0.30
0.75
0.32
0.63
0.34
0.83
0.34
0.33
0.31
500.0
1000.0
5000.0
10000.0
Limit
0.8
0.7
0.6
0.5
0.4


Topic Load Time (s)
Repartition Size
5000
1000
100
10
0.07
0.10
0.08
0.07
0.09
0.08
0.10
0.08
0.09
0.10
0.07
0.07
0.12
0.09
0.08
0.07
500.0
1000.0
5000.0
10000.0
Limit
0.115
0.110
0.105
0.100
0.095
0.090
0.085
0.080
0.075


Sentiment Compute Time (s)
Repartition Size
5000
1000
100
10
0.47
0.48
0.42
0.43
0.46
0.42
0.39
0.41
0.44
0.53
0.36
0.35
0.53
0.58
0.42
0.41
500.0
1000.0
5000.0
10000.0
Limit
0.575
0.550
0.525
0.500
0.475
0.450
0.425
0.400
0.375


Plot Time (s)
Repartition Size
5000
1000
100
10
72.32
83.91
266.44
396.11
63.99
102.47
243.80
366.98
58.35
71.01
202.27
298.98
59.20
77.60
179.03
339.79
350
300
250
200
150
100
500.0
1000.0
5000.0
10000.0
Limit


Total Time (s)
Repartition Size
5000
1000
100
10
86.66
112.90
285.88
429.06
78.65
116.50
281.27
382.21
73.66
85.20
216.07
312.92
77.69
92.20
193.11
374.20
400
350
300
250
200
150
100
500.0
1000.0
5000.0
10000.0
Limit



(Title: Total Pipeline Runtime (s) as Dataset Limit Size Increases)
(X-Axis: Limit Size / Y-Axis: Total Pipeline Runtime (s) / Legend: Repartition Size)

Works Cited

- https://github.com/fnielsen/afinn
- https://www.kaggle.com/datasets/mateibejan/15000-gutenberg-books?resource=download
- https://github.com/UniversalDependencies/UD_English-EWT