Rhyme-Viz

CSE6242 Fall 2017

Majid Ahadi Dolatsara
mad6@gatech.edu
PJ Fleming
pjfleming@gatech.edu
Jag Gangemi
jg5813@gmail.com
Robert Martin
rmartin95@gatech.edu

Introduction:

Writing a poem is a very rewarding and fun experience, and one of the steps to write one is finding words that rhyme together. This is one of the features that makes a poem different from daily conversation [1]. However, finding such words is not always easy. It can be time consuming and sometimes impedes the writers to articulate the true message that they had in mind. Of course, thesauri and rhyming dictionaries can be used. Nevertheless, these resources are fraught with words. There is usually no hint on what concepts or location in the sentence these words can be used. Therefore, the task of finding rhyming words remains laborious. Hence, we were encouraged to develop a program to help with the task.

This paper presents a report on the Rhyme-Viz project. Rhyme-Viz is a visual and interactive software that simplifies exploration of rhymes and synonyms in a graph format. This tool offers two solutions for finding rhymes. One is simply presenting rhyming words with the input word. The other is showing synonyms of the input word, accompanied by the rhyming words of each synonym. This helps the writer to replace the input word with another one that has more rhymes or is just more suitable for their poem. This feature can also be used as a simple thesaurus. From there, the user can use an explorer mode and hop from one word to another one in the graph to load new neighbors and discover all sort of new words.

In addition, Rhyme-Viz takes advantage of a database of top song lyrics in different genres in the past 20 years. Therefore, it suggests the genre in which a word is more common. Moreover, definition, antonyms and parts of speech are presented to help the writer with choosing a word. Rhyme-Viz benefits from visual elements like color, location, size and tooltips to relay this information.

To the best knowledge of authors, there is no software currently developed with similar capabilities. The closest thing is [2], which is basically a visual dictionary exclusive for iPads and not useful for finding rhymes.

Rhyme-Viz takes advantage of machine learning and natural language processing (NLP) techniques. We have studied similar techniques in the literature, for instance, [3] implements a system to determine “topic” and “focus” of each query. Reference [4] suggests a technique to measure semantic similarity. In [5] a crowd-sourcing method to classify words and language is presented. References [6] and [7] use Markov-Chain and an all-inclusive supervised learning algorithm, respectively, for tagging parts of speech and text analysis. A broader overview of NLP is presented in [8]. A method for detecting rhyming word is suggested in [9]; however, it does not have visualization. In Rhyme-Viz visualization is inspired by graph presentation techniques introduced in the class and literature [10], [11]. For development Python is used because of its available libraries and built in functions in language processing [12], and D3 is used for visualization because of its customizability and flexibility [13].

The rest of paper is as follows. Section II explains the proposed approach. It includes the data resources, cloud architecture and the visualization. Section III discusses evaluation techniques and experiments. Conclusion is presented in section IV.

Proposed method

In comparison with online websites and other approaches Rhyme-Viz is advantageous in different ways which will be explained in coming sections. And the development of Rhyme-Viz includes back-end algorithms and NLP, Visualization and user interface, and the cloud architecture. These sections are described in the following.

Algorithms - Natural Language Processing

We used a variety of Natural Language Processing techniques to build the word set for RhymeViz. The Carnegie Mellon Pronouncing Dictionary (extracted via Python NLTK) formed the basis of our master word list. We used the Python "pronouncing" API to build rhymes from our master word list. We used WordNet to build synonyms and antonyms from our master word list. We also pulled parts of speech and definitions from WordNet. The various data pulled from these APIs was compiled and synthesized into a JSON formatted text file, which was used as the basis for loading our database.

Visualization and User Interface

The final version of Rhyme-Viz's visualizations delicately displays relationships between words using color, scroll over, and node connections. The visualization consists of the inputted word at the center of the visualization with synonyms surrounding the inputted word in a graph like structure. The entered word is at the center of the visualization with synonyms, rhymes, and

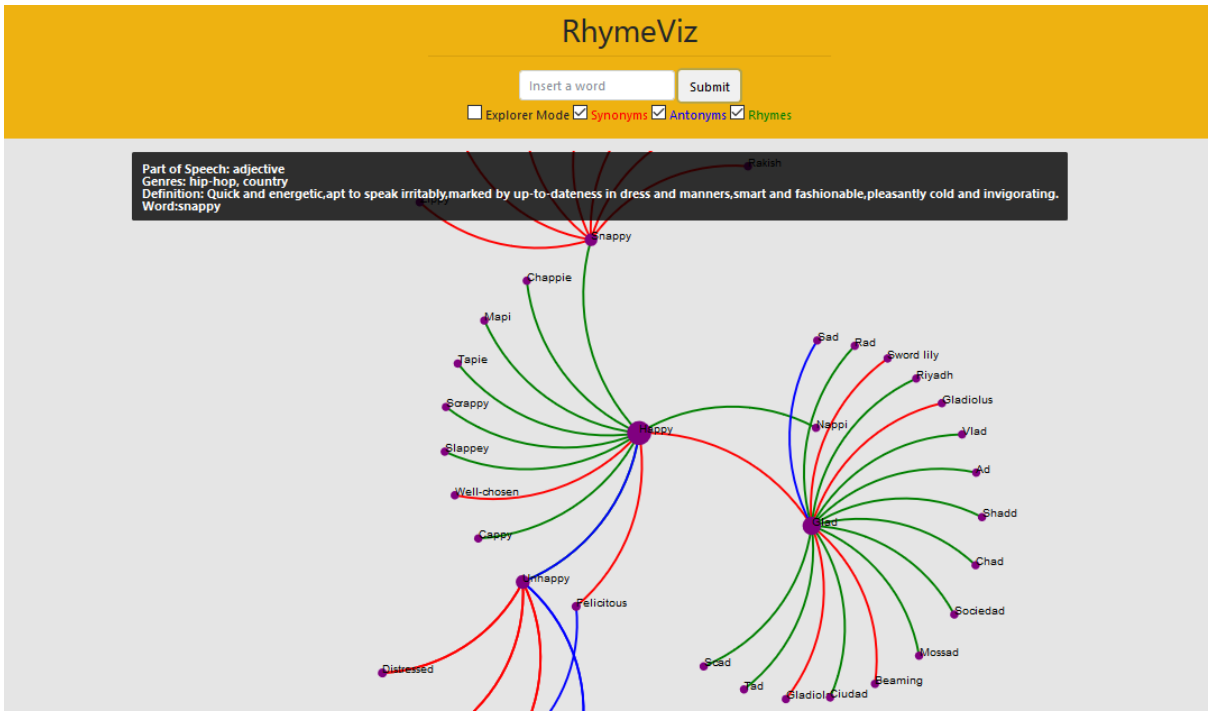

Figure 1 Rhyme-Viz user interface

antonyms around the root node. Words around the root node can be double clicked to display their graph structure visualization if the Explorer Mode is selected. The purpose of this structure is for the end user to input a word and receive an output that rhymes to a synonym of the inputted word to help writers block. The visualization now pulls synonyms, antonyms and rhymes from the group created application programming interface in JSON form using jQuery and visualizes the words as nodes with relationships as colored lines using D3.

The process for gathering the inputted word data involves standing up our own application programming interface and requesting the data for words through a get request with jQuery. The structured JSON format makes the data easy to store and search, thus creating an easy request system for the visualization. The request did not slow down when we expanded the JSON file to hold definitions, parts of speech, rhymes, synonyms and antonyms. The data is then visualized using D3. The advantages of using D3 for the visualizations is the almost unlimited customizability using D3. Custom functions for the relationships of data can be

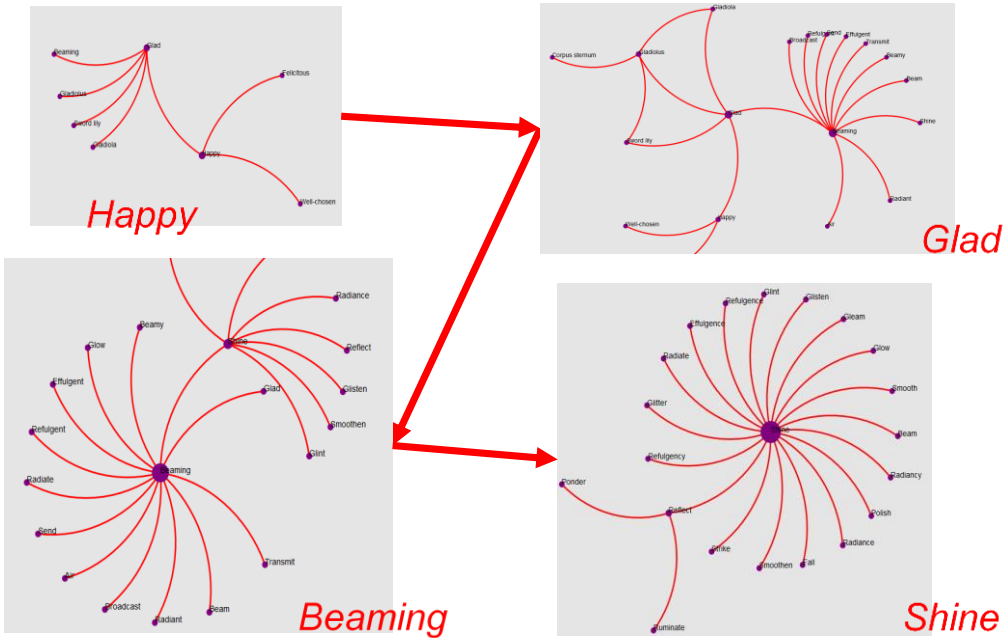

Figure 2 Explorer mode

made with D3 with several options for visualization, we have chosen to visualize the nodes and implement natural language processing to measure the correlation of the rhymes with the inputted word. Also, D3 is created to sustain visualizing even with very large amounts of data. The visualizations render almost instantly after receiving the information from the JSON request.

Once in the Rhyme-Viz web application there are major updates from the progress report section of the project schedule. The window now fits the whole page and is styled with CSS to clean up the input and Title section of the page. The title section now includes Explorer Mode, Synonyms, Antonyms and Rhymes checkboxes. The color of the relationship of the word

correlates with the color of the relationship line on the graph. The Explorer Mode option toggles the double click on nodes feature to change the root node, when selected, and to pin the double-clicked node, while not selected. The scroll over feature of the visualization gives important information on the hovered word. This information includes part of speech, applicable genres of music, and the definition. The applicable genres states which genre of music the word relates most to through data scraping the top hits in each genre at Billboard and Genius music websites. The overall structure and quick easy design makes Rhyme-Viz an easily navigable application for writers looking to solve writer's block.

Cloud Architecture

For the purposes of providing data to our user interface, we developed an API and database layer. To reduce the overhead of infrastructure management we decided to use “serverless” options for our architecture which run in Google Cloud Platform. Google manages the operations of all the infrastructure and at our scale we fall into the “free tier” which means there is no cost associated with the service. Nevertheless, our solution can scale up to millions of requests per day with no code changes. This is essentially the same solutions architecture that SnapChat has used to scale to billions of daily users.

The RhymeViz Cloud Architecture has the following components:

- **User Interface** - the user interface is implemented in HTML, D3, and JQuery. It is hosted in Google App Engine, which provides a “serverless” web runtime environment which will automatically scale to the number of incoming requests.
- **API Layer** - the API layer is implemented in Python and deployed to Google App Engine. The data protocol is REST and JSON. The API layer utilizes memcached to reduce or eliminate the need for API calls to the database.
- **Database Layer** - the database layer is implemented using Google Cloud Datastore, a “serverless” No-SQL key-value database. This database is extremely low latency (microseconds to low milliseconds) and automatically scales almost infinitely.
- **Data Ingest Layer** - the data ingest layer is implemented in Python running on Google Cloud DataLab - a managed iPython notebook service. The ingest layer processes a flat text file as its input and stores each row as a key-value pair.
- **External API Layer** - we utilize data from external APIs to supplement our natural language processing data. We call an unofficial Billboard API to download the top 300 songs in three genres across the last 20 years. Then we use the official Genius API to generate web URLs for lyrics. Finally, we screen scrape lyrics data from Genius HTML pages. Then we run word counts by genre in python to associate words with genres of music. The code is based upon this open source project. [14]

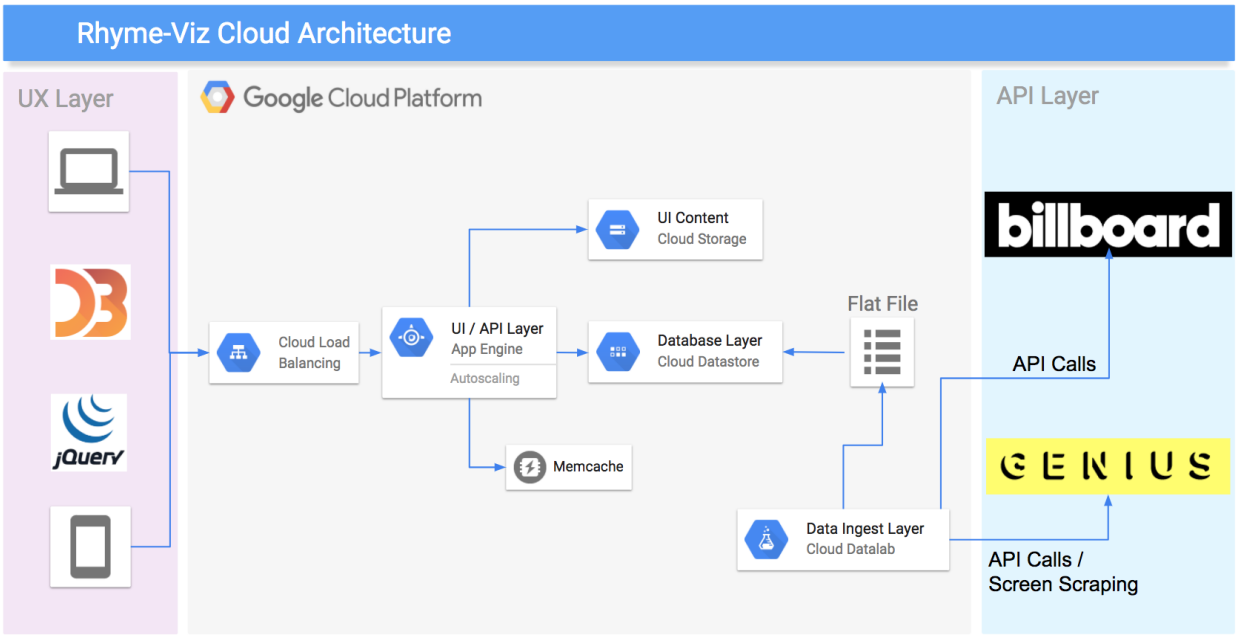

Figure 3. Rhyme-Viz Cloud Architecture

API Documentation

Request Word API - utilized to request a word from the database for display in the UI

Method: HTTP Get
URL: https://romartin-cse6242a.appspot.com/get
Parameter: word - the word being requested to load
Sample Request: https://romartin-cse6242a.appspot.com/get?word=happy
Response: JSON array
Sample Response:
[
{"genres": ["rock", "hip-hop", "country"], "word": "model", "pos": ["n", "v", "s"], "rhymes": ["podoll", "waddle",], "synonyms": ["exemplary", "mold"], "antonyms": [], "definitions": ["the act of representing something (usually on a smaller scale)"]}
,
{"genres": ["rock", "hip-hop", "country"], "word": "copy", "pos": ["n", "v"], "rhymes": ["groppy", "topkapi",], "synonyms": ["imitate", "re-create",], "antonyms": [], "definitions": ["reproduce or make an exact copy of", "a reproduction of a written record (e.g. of a legal or school record)"]}
]
Notes: The first element in the array will be the one requested. Remaining elements are words which are referenced in the “synonyms”, “antonyms”, or “rhymes” arrays.

Key Source Code Files

- **cse6242project/RhymeViz Vizualizations/src/RhymeVizV.2.4.html**
  - main user interface file
- **cse6242project/api/main.py**
  - main api file
- **cse6242project/db/synonyms.zip**
  - compressed synonyms2.json data file for loading the database
- **cse6242project/db/add_genres.py**
  - code to add genre data into base nlp dataset
- **cse6242project/db/genres/***
  - code and data files for music genre and lyric data
- **cse6242project/db/?** - code needed for NLP

Experiments/ Evaluation

Performance of Rhyme-Viz is evaluated through a user study, where it is compared with a thesaurus website (thesaurus.com) and a website for finding rhymes (rhymezone.come). These websites form the baseline of the experiment. In this Study Users were asked to the following first with Rhyme-Zone and then with the websites

1. Imagine you are writing a song
2. Choose an initial word and start a timer
3. Find a synonym for the initial word
4. Find a rhyme for the synonym and stop the timer
5. Explore and play with different options
6. Report timing, pros and cons and general feedback

Experiment Results

The user study conducted gathered information on the perceptions and speed of Rhyme-Viz from ten volunteers. This study consisted of asking users to test Rhyme-Viz versus current online tools for finding synonyms and rhymes. The user was asked to imagine that they are writing a song or poem and to choose a word that they want to swap out of their writing with a synonym and find a rhyme for the next verse. They ran some examples and timed themselves on the speed of finding a synonym then a rhyme of the synonym. The speed of Rhyme-Viz was similar to the use of current internet databases thesaurus.com and rhymezone.com. These results are presented in Table. 1.

Next, we asked follow up questions to the experiment. The questions gathered the overall perception of Rhyme-Viz as a program to the users. The perception of Rhyme-Viz over the internet alternatives was much higher. The users enjoyed seeing the relationships of words and explore the graph structure to find the perfect word combination. Users also enjoyed all relationships of a word being colorized and in one spot.

Table 1 Experiment results word sequences and times

User	Baseline websites sequence Input word -> Synonym -> Rhyme	Baseline time (s)	Rhyme-Viz sequence Input word -> Synonym -> Rhyme	Rhyme-Viz time (s)
1	Confuse -> Perplex -> Complex	27	Confuse -> Befuddle -> Muddle	54
1	Imagine -> scheme -> Dream	37	Imagine -> Guess -> Finesse	67
2	Atom -> Iota -> South Dakota	85	Atom -> Speck -> Beck	93
2	Cold -> Insensate -> horse sense	74	Cold-> Insensate -> -no rhyme-	73
3	Mad -> Sick -> slick	30	Mad -> Sick -> Slick	63
3	Doctor -> Physician -> Permission	20	Doctor -> Physician -> Permission	33
4	passion->mania->transylvania	44	passion->mania->transylvania	51
4	love->hump->jump	28	love->hump->jump	44
5	Happy -> glad -> scad	44	Happy -> glad -> rad	9
5	Maybe -> possibly -> cry baby	33	Maybe -> perhaps -> claps	21
6	Ride -> rally -> Sally	63	Ride -> sit -> fit	34
6	Summon -> cite -> write	46	Summon -> muster-> cluster	55
7	Taunt -> tease -> peas	43	Taunt -> rag -> tag	17
7	Bag -> purse -> disburse	45	Bag -> purse -> converse	26
8	Regretful -> sorry -> atari	27	Regretful -> sorry -> safari	33
8	Glad -> happy -> crappy	22	Glad -> happy -> chappie	16
9	Film -> movie -> groovy	41	Film -> flick -> click	13
9	Thick -> wide -> slide	49	Thick -> deep -> heap	37
10	Heavy -> thick -> sick	42	Heavy -> dense -> tense	26
10	Happy -> cheery -> theory	43	Happy -> glad -> tad	37

The feedback from test users is presented in Table 2. Although there were suggestions for improvements, in general they had positive experiences with Rhyme-Viz and supported the idea.

Table 2 Pros and cons of Rhyme-Viz

PROS	CONS
Cool design and idea	Sometimes it shows too many words
Shows relationships between words	Occasional software crash and bugs
Explorer mode makes searching easier	Limited synonyms
Genre analysis helpful for literature creation	Display window cuts off nodes, want to scroll in node window
All functionality in one program	Want a history tab to retrace previous words

Conclusions and Discussion

In this paper, a report on the Rhyme-Viz project is presented. Rhyme-Viz is an interactive assistant tool for poets. Innovations in this project are:

1- The interactive feature for finding rhymes and synonyms.
2- Huge online dataset and real time API.
3- Exploring related words and moving through the graph.
4- Using NLP to extract characteristics of each word.
5- Providing common genres, part of speech and definition for each word.

Work is distributed with roughly the same weight. Patrick and Majid focused on front-end visualization. Jag and Robert worked on the data base and back-end coding and NLP.

References:

[1] Strachan, John, and Terry, Richard. Poetry. Edinburgh: Edinburgh University Press, 2011, Chapter 3, pp. 47-71.

[2] Scheiner, Robert, “Wordflex touch dictionary.” Wordflex.com, http://www.wordflex.com, (Accessed October 1, 2017).

[3] Andersson, Marta, Adnan Ozturel, and Silvia Pareti. "Annotating Topic Development in Information Seeking Queries." In LREC. 2016.

[4] Agirre, Eneko, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. "A study on similarity and relatedness using distributional and wordnet-based approaches." In Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 19-27. Association for Computational Linguistics, 2009.

[5] Chang, Nancy, Russell Lee-Goldman, and Michael Tseng. "Linguistic Wisdom from the Crowd." In Third AAAI Conference on Human Computation and Crowdsourcing. 2016.

[6] Church, Kenneth Ward. "A stochastic parts program and noun phrase parser for unrestricted text." In Proceedings of the second conference on Applied natural language processing, pp. 136-143. Association for Computational Linguistics, 1988.

[7] Collobert, Ronan, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. "Natural language processing (almost) from scratch." Journal of Machine Learning Research 12, no. Aug (2011): 2493-2537.

[8] Chowdhury, Gobinda G. "Natural language processing." Annual review of information science and technology 37, no. 1 (2003): 51-89.

[9] Byrd, Roy J., and Martin S. Chodorow. "Using an on-line dictionary to find rhyming words and pronunciations for unknown words." In Proceedings of the 23rd annual meeting on Association for Computational Linguistics, pp. 277-283. Association for Computational Linguistics, 1985.

[10] Herman, Ivan, Guy Melançon, and M. Scott Marshall. "Graph visualization and navigation in information visualization: A survey." IEEE Transactions on visualization and computer graphics 6, no. 1 (2000): 24-43.

[11] Borup, Rick. “Data Visualization for the Database Developer.” Paper presented at the Southwest Fox conference, Gilbert, AZ, October 2015.

[12] Bird, Steven, Ewan Klein, and Edward Loper. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.", 2009. pp.1-21
[13] Murray, Scott. Interactive Data Visualization for the Web: An Introduction to Designing with. " O'Reilly Media, Inc.", 2017.
[14] https://github.com/rohankshir/lyrics_scraper