**Progress report on a comprehensive visual and interactive assistant tool to simplify exploration of rhymes and synonyms for poets (Rhyme-Viz)**

**I-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 categorization based on concept or part of speech. 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 progress report on the Rhyme-Viz project, which 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 a word that has more rhymes or is just more suitable for their poem. This feature can also be used as a simple thesaurus. In addition, clustering based on the meaning, part of speech and other characteristics are implemented in Rhyme-Viz by taking advantage of visual elements like color, location and size.

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.

To offer more relevant rhymes, Rhyme-Viz takes advantage of machine learning and natural language processing (NLP) 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 suggests evaluation techniques and activity plans. Conclusion is presented in section IV.

**II- Proposed approach**

1. **Data resources**

The functional aspects of RhymeViz are dependent on the integration of the visualization frontend of the application with our proprietary API. The application submits get requests through jquery, and our API returns JSON formatted data on words and their relationships. The frontend then displays the appropriate visualization for the data.

We chose to create and populate our own API to circumvent the challenges and expenses (and particularly rate limits) associated with get requests to foreign APIs. One of the key difficulties in this approach is populating the API with data; while synonym and rhyme lists exist in rhyme dictionaries and thesauruses, inferring them through natural language processing is a nontrivial task. To begin with, we have taken data directly from existing sources in the Natural Language Toolkit python library (primarily WordNet), along with other public-access databases of words with synonyms. By running some simple python programs, we’ve could extract data from these differently formatted data structures and place them into a standardized JSON format which uses the word as a key for the get requests. Since words, dictionaries, and thesauruses tend to be based on free, public data, we expect to be able to format nearly the entire English language as JSON objects with synonyms and other attributes. The result is a database populated with a heavily connected network of nodes, which can then be visualized by the frontend D3 programs for the user.

We chose JSON formatting simply due to its ubiquity; it’s applicable on nearly every platform and across coding languages, and integrates seamlessly with jquery and consumes minimal storage space. Additionally, since our concept is designed to be scalable across multiple attributes, we plan to add more types of edges to our network (such as antonyms, words which commonly appear together, and anagrams), and the flexibility and mutability of the JSON data model allows us to very easily modify our data network and to create simple interfaces for adding and changing data. Now we can begin using natural language processing to make word recommendations based on the data in our API, and then mapping those recommendations through the D3 visualization on the frontend.

1. **Cloud Architecture, API Layer, and Database**

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.
* **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.

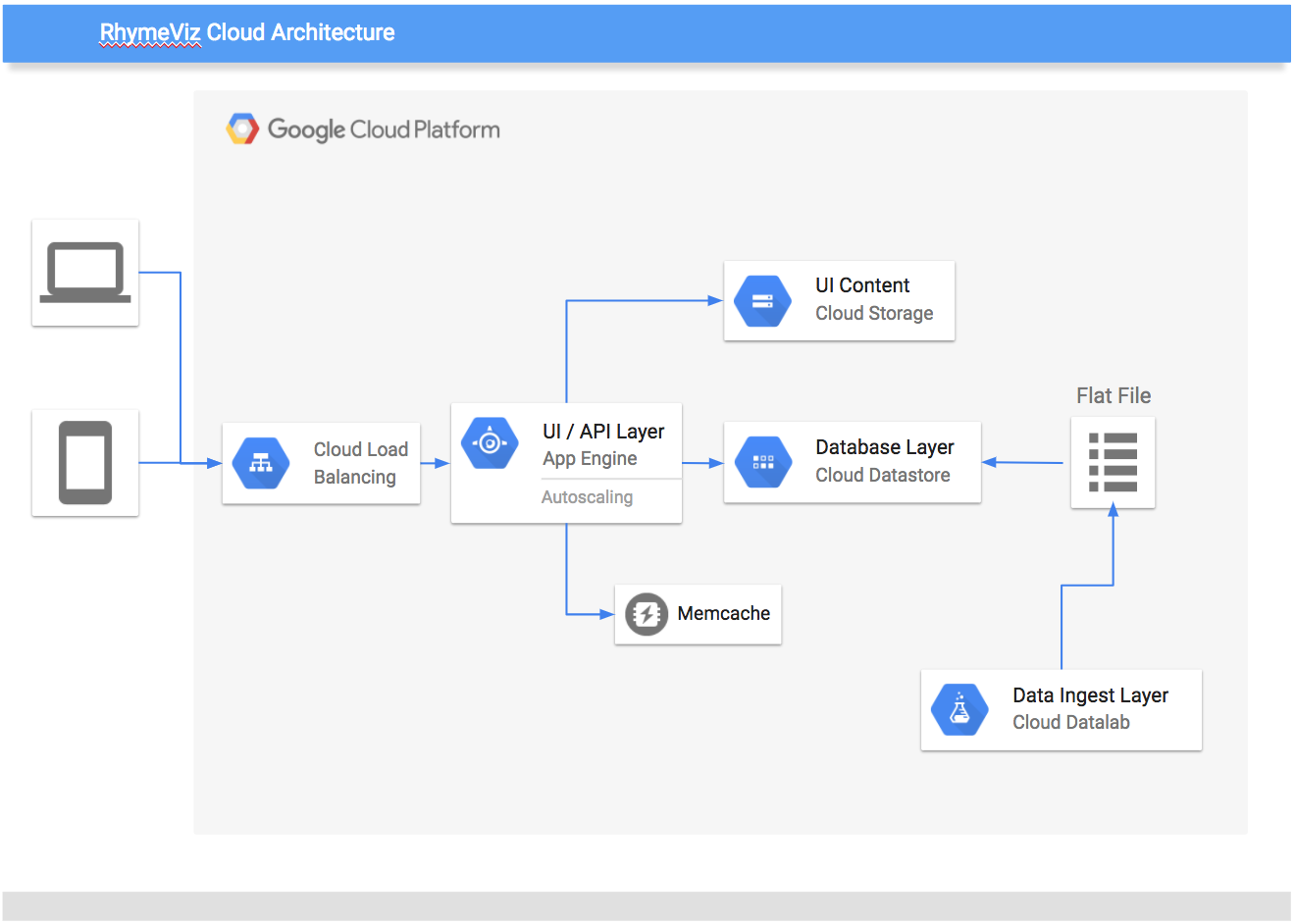


Figure 1. iGoogle Cloud Datastore for consumption by the API layer

1. **Visualization**

The preliminary visualization of RhymeViz is functional with synonym visualization and a toy dataset. The visualization consists of the input word at the center of the visualization with synonyms surrounding the input word in a tree like structure. The visualization now pulls synonyms from the group created application programming interface in JSON form using jQuery and visualizes the words as nodes with relationships as lines using D3.

The process for gathering the input 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 data is then visualized using D3. Custom functions for the relationships of data can be made with D3 with several options for visualization, we implement natural language processing to measure the correlation of the rhymes with the input word. Also, D3 is created to sustain visualizing even with very large amounts of data.

RhymeViz currently has the infrastructure and data connections made to visualize more with minimal additions. For the final product of RhymeViz we will visualize the rhymes around the synonym along with the relevance of the rhymes to the inputted word through natural language processing. The foundations for these visualizations have currently been made through setting up the application programming interface, data requesting in the visualization, and node relationship visualization. Furthermore, another addition to the visualization is specializing data sets for potential end users. (classic, romantic, hip hop, etc.)

**III- Evaluation**

Due to nature of this project the evaluation would be mainly visual and through user experience. Firstly we complete and improve the proposed approach in section II and making sure all sections are connected correctly. Currently a limited set of words are used which go through the API layer and appear on the screen after visualization. This is shown in Fig. 2. This is a simple presentation, nonetheless the final product would be a more complicated graph, where nodes and edges have different sizes and colors, and rhymes are included.

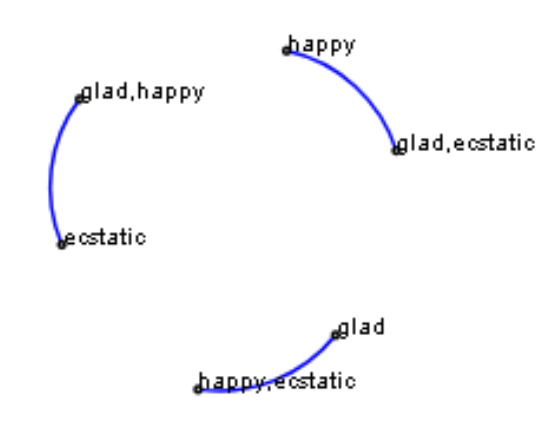


Figure 2

Revised plan of activities is shown in Table 1. We are currently at the end of week 3.

Table 1

|  |  |
| --- | --- |
| Week | Activity |
| 1 | Background studies and research |
| 2 | Setting up the data base and data format |
| 3 | Completing data resources and basic visualization |
| 4 | Code works for synonyms |
| 5 | Code works for Rhymes |
| 6 | Testing and evaluation |

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

**IV- Conclusion**

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

* 1. The interactive feature for finding rhymes and synonyms.
  2. Huge online dataset and real time API.
  3. Using NLP and machine learning to determine relevance of rhymes.
  4. Clustering of rhymes based on concept, part of speech, etc.

**Appendix A:** Old plan of activities:

The project will take 8 weeks. Three weeks focuses on language processing and the next three weeks focuses on visualization. The final two weeks is for testing. Midterm check for success will be testing natural language processing features. Final check for success will be on visualization testing.

**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.