# Famous Quotes Classification

Haoyang Zhou, Andrew Cui, Yuanzhen Zhang
Santa Clara University, Santa Clara, CA 95053, United States
hzhou3@scu.edu, zcui2@scu.edu, yzhang18@scu.edu

**Abstract.** This project uses the Beautiful Soup library to extract quotes from a popular quotes website, and classify them based on their content. The goal is to develop a model that can accurately categorize quotes into different categories, such as love, inspiration, humor, and others. The project will involve data scraping, data processing, and data retrieving. The resulting model could be useful for various applications, such as generating personalized quotes, recommending quotes based on user preferences, or analyzing trends in popular quotes. Overall, this project demonstrates the power of web scraping and machine learning in extracting useful insights from unstructured data on the web.

**Keywords:** Data Parsing, Machine Learning, Data Classification, Data Analysis, Natural Language Processing.

## 1. Introduction

In today's digital era, the amount of text-based data is growing exponentially. With the vast amount of information available online, it can be challenging to manually sort and retrieve it. To resolve this problem, various machine-learning techniques have been implemented, including text classification and retrieval. These techniques are beneficial for analyzing and extracting relevant information from large collections of textual data.
The Beautiful Soup library is a popular Python library for web scraping, data parsing, and HTML/XML processing. It provides a simple interface for traversing HTML and XML documents, and its flexible syntax makes it a powerful tool for extracting relevant information from web pages. In this research, we will use the Beautiful Soup library to

extract quotes from the web page (https://quotes.toscrape.com/) and classify them based on their attributes. We will explore various text classification techniques, including term frequency–inverse document frequency (TF-IDF) and Cosine similarity, to categorize the quotes based on their content. Finally, we will develop a retrieval system that can search for relevant quotes based on specific search criteria.

In Section 2, we will provide an overview of data sources (webpage). In Section 3, we describe the methodology (libraries) and data classification and retrieval technics used in this research. In Section 4, we present the results of our experiments, and in Section 5, we discuss the implications and limitations of our approach. Finally, we conclude the paper in Section 6.

## 2. Data Source

The webpage https://quotes.toscrape.com/ contains a collection of quotes from various authors. The format of the webpage is HTML and it contains multiple HTML elements, including headers, paragraphs, and list elements. The quotes are displayed in a structured format, with each quote enclosed within a div element that has the class "quote". The quotes we are interested in are enclosed under the class “text,” and the authors' names are under the class “author.” Thus, when applying data parsing, we would be using each class name accordingly.

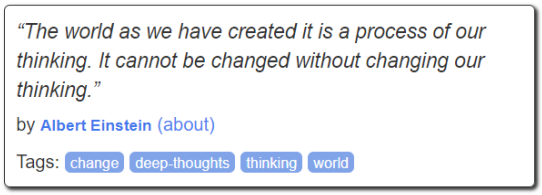


**Fig. 1.** Quote Sample.

Figure 1 shows the demonstration of a quote from the webpage. The quote box can be broken up into three sections, the quote, the author, and the tag.

```
<div class="col-md-8">
  <div class="quote" itemscope itemtype="http://schema.org/CreativeWork">
    <span class="text" itemprop="text">
      "“The world as we have created it is a process of our thinking. It cannot be changed without
      changing our thinking.”"
    </span>
    <span>
      "by "
      <small class="author" itemprop="author">Albert Einstein</small>
      <a href="/author/Albert-Einstein">(about)</a>
    </span>
    <div class="tags">
      " Tags: "
      <meta class="keywords" itemprop="keywords" content="change,deep-thoughts,thinking,world">
      <a class="tag" href="/tag/change/page/1/">change</a>
      <a class="tag" href="/tag/deep-thoughts/page/1/">deep-thoughts</a>
      <a class="tag" href="/tag/thinking/page/1/">thinking</a>
      <a class="tag" href="/tag/world/page/1/">world</a>
    </div>
  </div>
```

**Fig. 2.** HTML Sample.

Figure 2 shows the HTML source code of the quote box from figure 1. It is important to note that the quote, author, and tag elements are hard-coded.

# 3. Methodology

## 3.1 Data Parsing and Collection

To realize data parsing for the webpage, the Beautiful Soup Library will be used. According to the official Beautiful Soup Documentation, the library has the following desirable functions that will facilitate the research:

- Parsing: Beautiful Soup can parse HTML documents and create a parse tree that can be navigated and searched.
- Navigation: Beautiful Soup provides methods for navigating the tree, such as finding all elements with a particular tag or class name.
- Searching: Beautiful Soup provides powerful search capabilities, allowing users to search for elements based on their attributes or contents.
- Modification: Beautiful Soup also provides tools for modifying the parse tree, such as adding or removing elements.
- Encoding handling: It can handle various types of encoding while parsing and converting them to Unicode.

For the reasons above, the Beautiful Soup library would be the best option for the research (Richardson).

### 3.2 Data Processing Methods

To classify collected quotes and find the most suitable content, the research will be examining the term frequency–inverse document frequency (TF-IDF). To retrieve documents and rank the documents based on their relevance to the query, we will use the Cosine Similarity method.

The TF-IDF method is a statistical measure used to evaluate how important a word is to a document in a collection (corpus). TF stands for term frequency, which measures how often a word occurs in a document; IDF stands for inverse document frequency, which measures how rare a word (topic) is in the corpus. Finally, the TF-IDF score–the product of TF and IDF–explains how representative a word is to a document compared to the rest of the documents in the collection (Scikit).

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}, \qquad idf\,(t, D) = log\,( \frac{N}{count\,(d \in D{:}t \in d)} )$$

**Formula 1:** TF-IDF

With the TF-IDF score, we are able to generate tags that represent each quote and therefore can retrieve corresponding quotes when these tags appear in a query.

The Cosine similarity is used to measure the similarity between two text documents (note that a query can also be treated as a document). Measuring the level of similarity is essential in natural language processing, information retrieval, and machine learning, because it can help to identify relevant information and make accurate predictions based on similarities between documents or sentences (Scikit).

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||}$$

**Formula 2:** Cosine Similarity

With the Cosine Similarity, we would be able to provide a ranking to the retrieved documents, allowing the user to access the most relevant information they acquired.

# 4. Procedures

To achieve the research goal, we plan out the procedures as follows:

1. Scrape down data from the website using the Beautiful Soup library
2. Apply data processing to calculate the TF-IDF and Cosine Similarity
3. Realize two searching functions:
    a. Search by tag: return sentences that are tagged with query tag, with their ranks and the values of the cosine similarity.
    b. Search by token: return sentences whose contents are highly correlated with the query term, with their ranks and values of cosine similarity.

## **4.1** Data Scraping

To gather data from the website, we utilized the Beautiful Soup library to find targeted content using various parameters. We noticed that there are in total of 10 pages, and information such as text (the quote per se), author, and tags are all contained within the class “quote.” To access that information, we implemented a loop that parses through all 10 pages, and for each page, we have an inner loop that loops through all the quotes and information contained on the page; Inside the inner loop, for each quote, we scraped the information of texts, the author, and tags, and stored them into two local dictionaries.

## **4.2** Data Processing

With the collected information, we then applied TF-IDF techniques. The purpose of this is to generate a matrix to calculate the cosine similarity between each quote and future user query. To store the TF-IDF score, we constructed two 3D matrices each for TF (tf) and TF-IDF (tf_idf) and a 2D matrix for the DF (df) score.

- With all values in the TF matrix initially being 0, we iterate through each document and update the corresponding token count. After the TF matrix was fully updated, we then applied normalization to each row for future calculation of TF-IDF.

- For the DF matrix, we iterate through all quotes and incremented token counts if the term is found in the document.
- Finally, for the TF-IDF matrix, we first converted DF into IDF and then combined it with the TF matrix to complete the final TF-IDF matrix.

**4.3** Search by Tag

To realize the search by tag feature, we utilized the Cosine Similarity function to apply ranking to each document. With user input, we first convert the query into TF-IDF form. We constructed a document vector for the query and used the rows of the TF-IDF matrix for the documents as their vector. With the vector, we calculated the cosine similarity between the query and each document and stored the document and the score in a temporary dictionary (similarity). We then applied the sorting algorithm to sort the documents by Cosine Similarity, then display the top 10 documents and their Cosine Similarity scores.

**4.4** Search by Token

To realize the search-by-token feature, we treated the tags as the tokens in the document and applied the same technique of calculating the TF-IDF and cosine similarity as described in sections 4.2 and 4.3 to establish the notion of ranking. With the user input, the function prints out the top 10 documents that are highly correlated with the queried token along with the Cosine Similarity scores.

# 5. Results

## **5.1** Search by Tag

```
results = search_by_tags("life and love")
for s in results:
    print(s[0][:80], ': ', str(s[1] * 100)[:7] + '%')
```

```
It is better to be hated for what you are than to be loved for what you are not. :  99.9999%
But better to get hurt by the truth than comforted with a lie. :  72.0063%
You may not be her first, her last, or her only. She loved before she may love a :  69.3908%
The real lover is the man who can thrill you by kissing your forehead or smiling :  69.3908%
Love does not begin and end the way we seem to think it does. Love is a battle,  :  69.3908%
To love at all is to be vulnerable. Love anything and your heart will be wrung a :  69.3908%
There is nothing I would not do for those who are really my friends. I have no n :  38.7951%
If I had a flower for every time I thought of you...I could walk through my gard :  38.7951%
This life is what you make it. No matter what, you're going to mess up sometimes :  38.2906%
Life is like riding a bicycle. To keep your balance, you must keep moving. :  37.5341%
```

**Result 1:** Search by tags

In this result, we searched the tag “life and love” as a query for each document’s tags. We then applied it to the function “search_by_tags” to calculate the cosine similarity by converting it to the appropriate TF-IDF values first. The function displays the top 10 similar quotes that have related tags sorted in descending order. It is obvious that the first result quote is what we require which has a similarity of approximately 100%.

## **5.2** Search by Token

```
results = search_by_tokens("life and love")
for s in results:
    print(s[0][:80], ': ', str(s[1] * 100)[:7] + '%')
```

```
Good friends, good books, and a sleepy conscience: this is the ideal life. :  22.2260%
Love does not begin and end the way we seem to think it does. Love is a battle,  :  16.4617%
A lady's imagination is very rapid; it jumps from admiration to love, from love  :  14.6316%
All you need is love. But a little chocolate now and then doesn't hurt. :  14.5046%
The fear of death follows from the fear of life. A man who lives fully is prepar :  13.1928%
I'm the one that's got to die when it's time for me to die, so let me live my li :  13.0297%
Do not pity the dead, Harry. Pity the living, and, above all those who live with :  12.8439%
If you judge people, you have no time to love them. :  12.7675%
There are only two ways to live your life. One is as though nothing is a miracle :  12.0112%
A wise girl kisses but doesn't love, listens but doesn't believe, and leaves bef :  10.9680%
```

**Result 2:** Search by tokens

The function “search_by_tokens” is to search which quote has the most similarity with the query. In this result, our query is “life and love” as well, but we got a completely different result, which means the “serach_by_token” function and the “search_by_tag” function search in different repositories. We noticed that the cosine similarity percentage is relatively low in all the result quotes. We conclude that this could be due to our query is not highly related to any quotes from the webpage.

# 6. Implications and Limitations

## **6.1** Implication

Our research offers a way to optimize the search by ranking documents according to their relevance to a query, which would make searching more efficient. Many of the current searching methods utilize techniques such as exact matching, Jaccard similarity, and Euclidean distance, which are less efficient than the cosine similarity approach. Our research findings can be used to optimize existing search engines, allowing for more accurate information retrieval in a shorter period of time.

Another benefit of our approach is that the program can be easily adapted to other webpages. By adjusting function parameters, Our program can adapt well to any web page and helps researchers collect and extract the information they want.

## **6.2** Limitation

One limitation we need to solve is that it takes a long time to analyze and generate results when the data is large, which is especially time-consuming and difficult to apply if the user wants to take turns searching for different keywords. We have brought up several ways to solve this problem:

- Use more efficient scraping libraries to shorten the time for calculating and extracting data.
- Optimize the scraping code by using asynchronous programming, caching, or parallel processing to enhance the performance.
- Use powerful data processing libraries such as Pandas or Apache Spark to process the data more quickly.

Another limitation is that it is hard to apply the Beautiful Soup library to other data sets since it is used to extract data from web pages specifically. Therefore, our model can only be used to apply on other web pages which are based on HTML and the information is hard coded.

## 7. Conclusions

To conclude, we built a model to help users to find the most similar quotes with keywords or tags they provided. Using the Beautiful Soup library, we extract the quotes and tags from HTML web pages, then process these data through “search_by_tag” and “search_by_token” using TF-IDF and cosine similarity techniques. When the user inputs a query, the functions will compare them with the web pages’ data. After the calculation, the top 10 results are displayed to users in descending cosine similarity order. Our findings demonstrate the efficiency of web and data mining techniques in collecting and retrieving information in the world of big data.

## 8. Work Cited

“Quotes to Scrape.” *Quotes to Scrape*, Scrapinghub, https://quotes.toscrape.com/.

Richardson, Leonard. “Beautiful Soup Documentation.” *Beautiful Soup Documentation - Beautiful Soup 4.9.0 Documentation*, https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

“Sklearn.feature_extraction.Text.TfidfVectorizer.” *Scikit*, https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

“Sklearn.metrics.pairwise.cosine_similarity.” *Scikit*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html.