Spring 2022

Identifying The Relationship Between Page Content and Title

Yabo Ornella Detchou

Bard College

Recommended Citation

Detchou, Yabo Ornella, "Identifying The Relationship Between Page Content and Title" (2022). *Senior Projects Spring 2022*. 126.
https://digitalcommons.bard.edu/senproj_s2022/126

Identifying The Relationship Between Page Content and Title

Senior Project Submitted to
The Division of Science, Math, and Computing
of Bard College

by
Yabo Detchou

Annandale-on-Hudson, New York
May 2022

Dedication

To my family

Acknowledgements

Thank you to the entire Computer Science department for their guidance and special thanks to my advisor for directing me on the road to completion.

Abstract

This project seeks to find the similarity score between content on the page and title using cosine similarity from a word2vec model. Frequent words and randomly chosen words from each article were analyzed and compared against the title using three samples. Frequent words were found to have a higher similarity score with the title than random words. Word frequency helps you identify the most relevant keyword on the page. The bigger goal of the project is to develop a keyword suggestion tool. Identifying which keywords are most relevant in writing content is the first step.

Table of Contents

Introduction

There are many steps involved in the keyword suggestion process. Keyword suggestion tools are designed to assist writers identify the ideal keywords to help them reach their target audience and rank well in search engine results pages. For my research, I will analyze Wikipedia articles and see if there exists a relationship between the top frequent terms on the page and the title of a page. The top frequent words tell you what the page is about and so does the title of the page. If I can prove that there is a similarity between the top frequent words and the title, then I have identified one factor that makes a keyword relevant is how frequently that word appears on the page.

Ranking

Search result ordering affects the likelihood of searchers clicking on your website as it appears in the search result page. Typically, searchers consider only the first several search results; the top-ranked search result receives a significant percentage of search clicks, and click-through rates rapidly fall from there (Goldman, 2018). This implies that searches typically ignore nearly all of the search results not on the first page. If your website is not on the first page, your chances of being seen by a user is minimal. Therefore, it becomes critical to find the best Search Engine Optimization (SEO) tool that will rank you among the top search results.

Search Engine Optimization

With bloggers' increased awareness of people-based marketing benefits from blogging, there is high demand for the best search engine optimization tools to help blog posts appear at the top of search engine results. In order to meet these demands, Chrome developers have focused on the development of new search engine optimization (SEO) techniques for improving visibly on Google. SEO techniques are classified into several approaches: structure optimization, which achieves a balanced website structure in various areas; keyword optimization- which primarily improves and emphasizes keywords and content optimization- which improves a site's subject matter by prioritizing content that will keep the site lively and eliminate content whose quality is low (Chotikitpat, Nisook, & Sodsee, 2015). The primary components of the SEO approach are improving and looking for the best keywords, or what the writers refer to as 'Golden Keywords' (i,e., niche keywords, misspellings, related keywords, prefix, suffix, etc.) (Chotikitpat et al., 2015). The techniques mentioned above, (structure optimization, keyword optimization, and content optimization), take into consideration SEO principles and best practices which emphasize the importance of keyword optimization as it naturally leads to the understanding of how to structure a website (Chotikitpat et al., 2015).

Bias

There are many problems with current SEO tools that keep blogging websites from ranking near the top of the search result page. One of these problems is "search engine bias," a phenomenon used to describe systematically promoting some content over others. Search engines are designed to attain the wants and needs of their audience, and they will go to great lengths to achieve this goal, even if that means the editor controls what you see (Goldman, 2018). To avoid anarchy and maintain credibility, search engines must exercise some editorial control over their systems. As a result of this editorial control, some prejudice will occur (Goldman, 2018). The problem with bias is that ranking algorithms determine the order of search results, and no one aside from the developer knows what factors are included in the ranking algorithm. In ‘Google Ranking Factors: What's The Secret Sauce?’ the authors mentioned that Google won’t reveal its top ranking factors (Forbes, 2021). This problem is important because only the top search results are clicked on (Goldman, 2018); thus, without understanding how the ranking algorithm works, your chances of having your website appear on the top search results and be clicked on are low.

Although biases in search engine optimization may be looked down upon, they are actually necessary and desirable for several reasons. From Goldman’s perspective, the unavoidable consequence of search engines exercising editorial control over their databases is search engine bias. Search engines, like any other media organization, simply cannot passively and neutrally spread third-party information (in this case, web publisher content) (Goldman, 2018). It is against the law. As stated in section 230 of the Title 47 of the United States Code, no one can legally publish or speak on information that is provided by someone else (Wikipedia Contributors, 2022). So search engine bias in this case is necessary to maintain the law. Furthermore, if third party content were easily accessible, the search engine system risks being engulfed by “spammers, fraudsters, and malcontents”(Goldman, 2018). This is called identity theft and it happens when scammers try to steal the identity of the original creator for illegal purposes. This can be avoided if the search engine undertakes action to hierarchize web content by removing third party information.

If a search engine cannot produce useful results, it no longer has a purpose. It makes sense then that those biases are desired to regulate the distribution of spam and valueless content to searchers. Another reason why biases are needed is that a search engine does not always get right what the searcher is looking for from only a few keywords. Searchers have high expectations of search engines: they want them to read their minds and deduce their purpose based on a limited number of search phrases. Search engines that disappoint (either by failing to produce relevant results or by burying useful results behind an excessive number of useless results) must answer to fickle searchers (Goldman, 2018). Many people fail to understand that search engines are playing a guessing game with our key terms because search engines don't know our intentions.

The Need to Serve the Searcher

Search engines are not people, they are machines, thus they cannot understand our sentiment, mood, and intent. In order for search engines to understand us, you [the developer] must add 'meaning,' which no calculating machine can accomplish because computers can only shuffle strings of binary numbers indefinitely, nothing more, nothing less (Bob Johnson, 2020). In order to find search intent behind a query, we must understand the browsing behavior because it tells us what the searcher is looking at which may have led them to search something.

Understanding User Browsing Behavior

“Real-world information retrieval (IR) heavily relies on effective usage of implicit feedback, which comes in various forms such as document clickthrough, viewing, scrolling, and bookmarking” (Lui, White, & Dumais, 2010). The benefit of studying user browsing behavior is that it helps us identify document relevance. Document relevance is important because “the standard approach to information retrieval system evaluation revolves around the notion of relevant and nonrelevant documents” (Manning, Raghavan, & Schütze, 2009).

Furthermore, according to Lui et al. (2010), many studies have looked into the relationships between implicit feedback and document relevance, and found that document dwell time (the amount of time a user spends on a document) is generally the most significant indicator of document relevance, aside from clickthrough, though the magnitude of the relationship varies depending on the information seeking task. The more time a user spends on a page, the more relevant that document becomes. As a result of the association between time and document relevance, dwell time has been effectively employed in a variety of applications, including learning to rank, query expansion, and determining query-independent page significance (Lui et al., 2010).

Search Classification

User studies from the Pennsylvania State University and the Queensland University of Technology tested a method for discovering user aims in web search engine queries by classifying user search based on their usage of words. The researchers used their classification system to automatically classify a different Web search engine transaction log containing over a million queries made by hundreds of thousands of users. According to their studies, "more than 80% of Web inquiries are <u>informative</u> in nature, with the remaining 10% being <u>navigational</u> or <u>transactional</u> in nature" (Jansen, Booth, & Spink, 2007). The general objective was to (1) isolate informative, navigational, and transactional aspects for Web searching queries by defining characteristics of each query type that will lead to real-world categorization. (2) Validate the taxonomy by automatically categorizing a large collection of Web search engine queries (Jansen et al., 2207). The three categories are navigational, transactional and informational searching. Navigation searchers want to find a website (Jansen et al., 2008). Transactional searchers want to find a product within a website, which may require executing some Web service on that Website (Jansen et al., 2008). Information searchers want to find information about a topic that will meet their needs (Jansen et al., 2008).

Navigational Searching

- Queries containing company/business/organization/people name
- Queries containing domains suffixes
- Queries with "web" as the source
- Queries length (i.e., number of terms in query) less than 3
- Searcher viewing the first search engine results page

Transactional Searching

- Queries containing terms related to movies, songs, lyrics, recipes, images, humor, and porn
- Queries with "obtaining" terms (e.g., lyrics, recipes, etc.)
- Queries with "download" terms (e.g., download, software, etc.)
- Queries relating to image, audio, or video collections

- Queries with “audio”, “images”, or “video” as the source
- Queries with “entertainment” terms (pictures, games, etc.)
- Queries with movies, songs, lyrics, images, and multimedia or compression file extensions (jpeg, zip, etc.)

Informational Searching

- Uses question words (i.e, “ways to”, “how to”, “what is”, etc.)
- Queries with natural language terms
- Queries containing informational terms (e.g., list, playlist, etc.)
- Queries that were beyond the first query submitted
- Queries where the searcher viewed multiple results pages
- Queries length (i.e., number of terms in a query) greater than 2 transactional

Result of each category and its characteristics

The conclusion of this research is that for search engines to become less reliant on humans as controllers, they must acquire the skill to understand user behavior, specifically what motivates searchers and their intent (Jansen et al., 2207). Thus, the approach of “automatically classifying queries” as coined by Jansen, Booth, and Spink (2007), is a feasible solution for search engines to arrange user intent based on the desired content. However, there are issues of accuracy with this method because the study classified each query to one and only one category, disregarding that a query may have multiple intents (Jansen, et al., 2007). Being able to identify all the possible intentions of a searcher can increase the performance of search engines and reduce the need for human input.

In a follow-up study, the authors utilized the same classification system. The research assumes that each search query has a goal that can be classified into one of these three classes, and if there exists ambiguity in the query, it will be grouped based on the highest connection of a category (Das et al., 2013). The focus of this research paper is on the ambiguity of key terms that do not fit into the three categories listed. The research applied fuzzy rules for classifying query goals in hopes to remove such ambiguity. There are two rules: the first rule divides clear objectives into three classes, while the second determines if the goal is ambiguous and which classes it is most likely to belong to (Das et al., 2013). The authors examine the dominant class for each query. If the dominating class receives considerably more clicks (votes) than its competitors, it is explicitly approved as the query objective. Otherwise, the query objective is expected to be unclear, and the dominating class and its nearest competitor(s) are given as the most likely query goals. (Das et al., 2013).

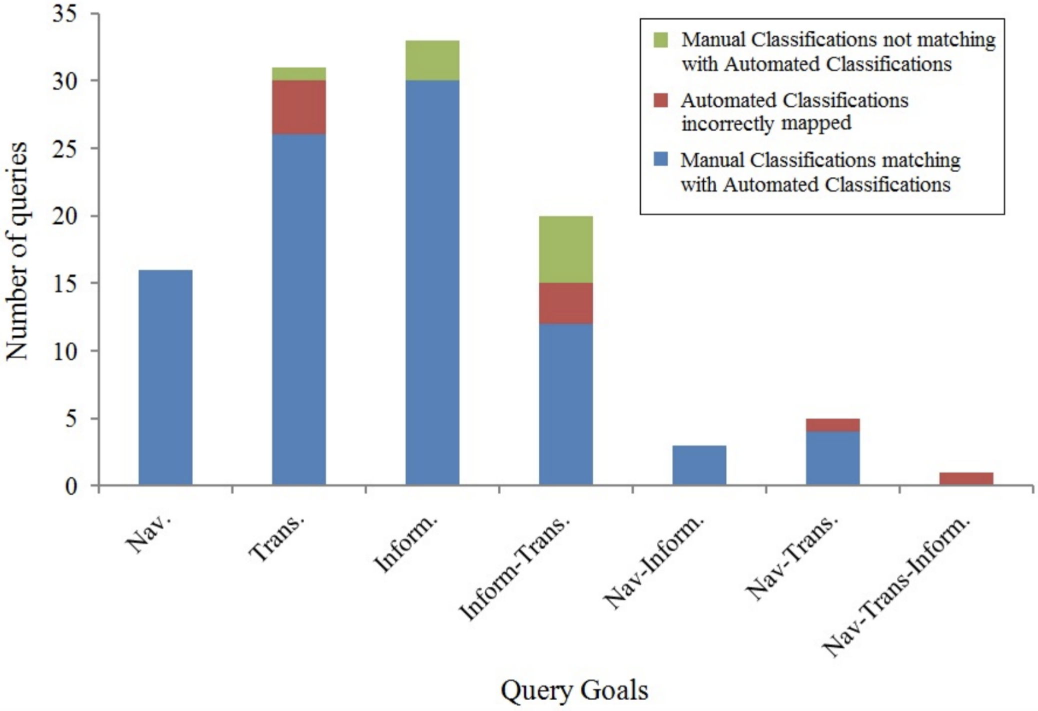


Figure 3 From (Das et al., 2013) showing Manual vs Automated Classification

The results show that there is a high number of queries that had manual classifications matching with automated classifications, which means that the proposed theory was a success. As the results reveal, the bulk of searches made to a search engine have predictable, unambiguous aims that can be detected to a large extent by their classier (Das et al., 2013).

One problem with current search engines is their ranking algorithms that are 'one-size-fits-all' in order to give uniform search results to searchers with diverse search objectives. One-size-fits-all algorithms increase the repercussions of search engine bias in two ways: (1) they generate winners (websites ranked high in search results) and losers (those with marginal placement), and (2) they give poor results for minority searchers (Goldman, 2018). The problem with using algorithms is that they aren't built according to the individual searcher's specifications or needs. Google, for example, provides users with the ability to "arrange your search results depending on your previous queries, as well as the search results and news articles you've clicked on. (Goldman, 2018). Thus, implementing a personalized approach to SEO tools will revolutionize the way searchers find information. Personalized algorithms go beyond those constraints, maximizing relevance for each searcher and so indirectly doing a better job of mind-reading searchers as well as reducing the consequences of search engine bias (Goldman, 2018). Goldman (2018) identified that one of the greatest benefits of the personalized algorithm method is that it limits the support given to "popularized based metrics (to give more weight for searcher-specific factors), reducing the structural biases due to popularity". An article from Ahrefs (2020), a popular SEO plug-in tool, came up with '26 Best Free Chrome Extensions for SEOs. They mentioned that there are different types of SEO plugins based on what they optimized. The different categories include On-page SEO, keyword research, Ranking checking, Technical SEO, Link building, All-in-One, and Miscellaneous. For this project, we will be implementing a personalized ranking algorithm and focusing on keyword research.

Keyword Research

By understanding the user's search intent, we can find the most relevant keywords for bloggers to target. In an ideal world, a company would produce online content around relevant keywords with large search volumes and minimal levels of competition. This would allow the company to rank high on the SERPs and collect a substantial portion of the enormous possible number of clicks (Nagpal & Petersen, 2020). The quickest way to rank on Google is by understanding the searcher's intent as we discussed earlier and by ranking first for a key term. For the purpose of my research, I did not implement an entire keyword suggestion tool but I have taken the first step in identifying how frequent words may be possible keywords because they are relevant to the content and the title.

Methods

Core of My Project

The core of my project is to measure and display the degree of similarity between frequent words versus random words against the page title, given a Wikipedia article. This is the core of my project because I want to find out which type of keywords a website is ranking for.

When writing an article, it is a good practice to include your keywords often because it sets the framework for your content. Including your keywords can help you rank for that keyword. Of course, it is impossible to know exactly which keyword your website will rank for because there are multiple factors involved in ranking but there are steps you can take to help you rank for a keyword. How your keyword ranks also depends on how you optimize for it. You want to leverage your keywords where you can in your content. Great places would include headings, subheadings, first sentences, etc.

Moreover, the title of a page tells you what the page is about and also sets the framework for your content. Thus, there should be a relationship between the words in the title and the most frequently used words.

If I can prove that the frequent words have a relationship to the title, then I can suggest that frequent words can be used as keywords to help you rank when writing content.

Why Wikipedia?

My dataset comes from the Wikipedia Python library. I used Wikipedia because the content is optimized very well and there is always structure that all Wikipedia articles follow and there should be consistency in where the keywords are placed on each article. Wikipedia articles always rank better than other websites especially for information queries. This is because Wikipedia articles have great, well written content.

Word2vec

My word vectors come from word2vec. Word2Vec is an approach used in Natural Language Processing to extract the relatedness across words. I used a word2vec model to measure the similarity between words and content on the page.

Word2vec Cosine Distance

I measured the cosine distance between the frequent word and the random words against the title. The similarity score was calculated using the gensim similarly function. I hypothesize that frequent words have a closer similarity to page title than random words.

Measuring Good Cosine Scores

The cosine similarity ranges between -1 and 1. If two vectors are perfectly the same, then their cosine score is 1. If two vectors are opposite, their similarity score is -1. If two vectors are perpendicular to each other, they have a similarity score of 0. To determine which variable has a closer relationship to the title, we need to find the higher frequency of cosine score of 1 or close to 1.

Algorithm for Comparing Words to Titles

I created one class, CosineSim. Within the class, I imported libraries to facilitate preprocessing, and data plotting. I have five methods: random_wiki_articles(), preprocessing(), get_similarity_score(), and histogram_plot().

1. random_wiki_articles()
    - Gets 500 articles from wikipedia API.
    - Extracts content and title and returns them in a list.
2. preprocessing()
    - Call the random_wiki_articles() method.
    - Performs sentence tokenization on the list of contents and the list of titles.
    - Call the collection module to use the counter method to find the frequency of each word. Take the first 25 frequent words using the most_common() method.
    - Call the shuffle method to shuffle the words in my content. Take 25 words.
    - Returns two tuples. One of the words in turtle and frequent words and another of the words in title and random words.
3. get_similarity_score()
    - Call the preprocessing() method.
    - Loads the word2vev model ‘word2vec-google-news-300’

    - Gets the cosine similarity for words in title with random words and with frequent words.
    - Calculates the percentage of words that are not found in the vocabulary.
    - Returns two arrays of cosine similarities scores. One for random words and the other for frequent words.

4. histogram_plot()
    - Call the get_similarity_score() method.
    - Plots a histogram of the frequency words against the title and the random word against the title.

Code Breakdown

def random_wiki_articles():

This method initializes a wikipedia article. Uses the matplotlib Pyplot library as opposed to other data plotting methods within python because matplotlib allows me to plot arrays, not just the entire dataset from wikipedia. Checks if a wikipedia article exists and retrieves a set number of random articles. Gets the title of each article along with their content and the pageid. Lowercases the titles and contents. Stores all articles and contents in a list and returns them.

```
@staticmethod
def random_wiki_articles():
    num_of_articles = 500
    generate_random_articles = wikipedia.random(num_of_articles)
    retrieve_titles = []
    retrieve_contents = []
    page_ids = []
    for articles in generate_random_articles:
        try:
            wikipedia.set_lang('en')
            wikipedia_page_object = wikipedia.WikipediaPage(title=articles,
                                                            pageid=True,
                                                            redirect=True,
                                                            preload=False,
                                                         original_title=u'')
            lowercase_titles = str(wikipedia_page_object.title).lower()
            lowercase_contents = str(wikipedia_page_object.html()).lower()
            cleantext = BeautifulSoup(lowercase_contents, "html").text
            page_ids.append(wikipedia_page_object.pageid)
            retrieve_contents.append(cleantext)
            retrieve_titles.append(lowercase_titles)
        except wikipedia.exceptions.DisambiguationError as e:
            e.options
        except wikipedia.exceptions.PageError as e:
            e
    print(page_ids)
    return retrieve_titles, retrieve_contents
```

def preprocessing():

This method starts with calling the random_wiki_articles() function that returns the list of titles and contents. The preprocessing steps include: word tokenization, removal of stop words, removal of punctuation for titles and contents. Then I use the parsing.preprocessing module from gensim to remove stop words. I remove punctuation and use nltk.tokenize to word tokenize. After the preprocessing steps are done, for my titles, I convert the word tokens into a dictionary. After preprocessing steps are done for my contents, I find the frequency of each word. I choose the top 25 frequent words and 25 randomly chosen words for analyzing frequent word vs random words against title. I create two dictionaries: one to store the random words and the other frequent words. I create a tuple and store the title token list and the list of random words as one pair. I create another tuple and store the title token list and the list of frequent words as one pair. I return the two tuples.

```
@staticmethod
   def preprocessing():
       titles_list, contents_list = CosineSim.random_wiki_articles()
       removes_stop_words_titles = []
       rv_st_content = []
       num_of_words = 25
       pre_processing_titles = dict()
       pre_processing_contents = dict()
       pre_processing_contents_random = dict()
       for titles in titles_list:
           titles = remove_stopwords(titles)
           removes_stop_words_titles.append(titles)
       for index in range(len(removes_stop_words_titles)):
           tokenized_titles = word_tokenize(removes_stop_words_titles[index])
           tokenized_titles = [word.lower() for word in tokenized_titles if
word.isalpha()]
           title_dictionary = dict.fromkeys(tokenized_titles, index)
           pre_processing_titles[index] = list(title_dictionary.keys())

       for content in contents_list:
           content = remove_stopwords(content)
           rv_st_content.append(content)
       for jindex in range(len(rv_st_content)):
           tokenized_contents = word_tokenize(rv_st_content[jindex])
           tokenized_contents = [word.lower() for word in tokenized_contents
if word.isalpha()]
           random.shuffle(tokenized_contents)
           content_dictionary_frequent =
dict(Counter(tokenized_contents).most_common(num_of_words))
           n = len(Counter(tokenized_contents))
           content_dictionary_random = tokenized_contents[:num_of_words] if n
> num_of_words else tokenized_contents
           pre_processing_contents[jindex] =
list(content_dictionary_frequent.keys())
           pre_processing_contents_random[jindex] =
list(content_dictionary_random)
           tuple_frequents = tuple(
               zip(list(pre_processing_titles.values()),
list(pre_processing_contents.values())))
```

```
            tuple_randoms = tuple(
                zip(list(pre_processing_titles.values()),
list(pre_processing_contents_random.values())))

        return tuple_frequents, tuple_randoms
```

def get_similarity_score():

This function calls the preprocessing() function that returns the two tuples. I import the gensim.downloader to load the word2vec-google-news-300. A for loop is created to loop through each key and value in my tuple lists and calculate similarity scores between each word of the title (keys) and each word in the frequent words and random words (values). I created two dictionaries to store the cosine similarities. If a value is not present in the dictionary, I append the value in a list. Thus, I have 2 lists for when value is not present for my frequent words and for when value is not present for my random words. This is done to calculate and print the percentage of words not present for frequent words and for random words. The function returns the two dictionaries containing the cosine similarity scores for the frequent words and for the random words.

```
@staticmethod
    def get_similarity_score():
        tuple_frequents, tuple_randoms = CosineSim.preprocessing()
        model = api.load('word2vec-google-news-300')
        cosine_sim_frequent_dictionary = {}
        cosine_sim_random_dictionary = {}
        values_not_present_frequent = []
        values_not_present_random = []
        total_values_frequent = []
        total_values_random = []

        for frequent_k, frequent_v in tuple_frequents:
            for fkey, fvalue in it.product(frequent_k, frequent_v):
                total_values_frequent.append(fvalue)
                try:
                    cosine_sim_frequent_dictionary[fvalue] =
model.similarity(fkey, fvalue)
                except KeyError:
                    values_not_present_frequent.append(fvalue)

        for random_k, random_v in tuple_randoms:
            for rkey, rvalue in it.product(random_k, random_v):
                total_values_random.append(rvalue)
                try:
                    cosine_sim_random_dictionary[rvalue] =
model.similarity(rkey, rvalue)
                except KeyError:
                    values_not_present_random.append(rvalue)

        percentage_frequency = (len(values_not_present_frequent) /
len(total_values_frequent)) * 100
        print(f"Percentage of words not present for frequent words is
{percentage_frequency} %")

        percentage_random = (len(values_not_present_random) /
len(total_values_random)) * 100
        print(f"Percentage of words not present for random words is
{percentage_random} %")

        return cosine_sim_frequent_dictionary, cosine_sim_random_dictionary
```

def histogram_plot():

This function calls the get_similarity_score() function that returns the two dictionaries containing cosine similarity scores for frequent words and for random words. I create two lists, one for frequent words, the other for random words, to store the scores to be used as x values. I create a final list, x_values, to store the two lists. x_values is the list containing the x-value used to create a single histogram that graphs the two arrays simultaneously. I create a label list that stores the name of my two variables. This label list is the legend to indicate which bar is which. I create the x and y labels and the title. It shows the grid in the y direction, and shows the edge colors. I create a bin for the weight of each bar. The bin represents the possible range of similarity scores. Finally, I plot the histogram using plt.show().

```
@staticmethod
   def histogram_plot():
       cosine_sim_frequent_dictionary, cosine_sim_random_dictionary = 
CosineSim.get_similarity_score()
       try:
           x_value_frequent = list(cosine_sim_frequent_dictionary.values())
           x_value_random = list(cosine_sim_random_dictionary.values())
           x_values =[x_value_frequent, x_value_random]
           labels = ['frequent words','random words']

       except statistics.StatisticsError:
           print("Data is empty")
       bins = [-0.4, -0.2, 0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
       plt.hist(x_values, bins=bins, label=labels, alpha=0.5, 
edgecolor='black')
       plt.legend(loc='best')
       plt.title('Similarity Score: Frequent words vs Random words')
       plt.ylabel('frequency')
       plt.xlabel('cosine scores')
       plt.grid(axis='y')
       plt.show()
```

Results

I created three analyses. Figure 1 shows my input size of 500 articles. The top 25 frequent words and 25 randomly chosen words were compared to each word in the title, and cosine scores were calculated. 43.5% of frequent words were not in my vocabulary, and 46.9% for random words. In contrast, Figure 2 shows 500 articles that were used as the input size, and the top 100 words were randomly picked and compared to each word in the title, yielding cosine scores. In this case 44.2% of frequent words were not in my vocabulary, and 44.1% for random words. Figure 3 shows my input size of 100 articles with the top 5 frequent words and 5 random words compared to each word in my title. The percentage of words not present was 47.8% for frequent words and 52.4% for random words.

Figures 1 and 2 show that the overall distribution of frequent words is not very different from the random words. However, I found that the percentage of the total words with a big similarity score goes upwards for my frequent words, but stays flat for my random words in both Figures 1 and 3. The pattern I noticed between Figure 1 and 3 is that they are comparing fewer words which implies that only the top 5-25 frequent words are identified as being similar to the title. Thus when we compare each cosine score for the frequent words and random words, the frequent words are higher frequency for the higher value of similarity.

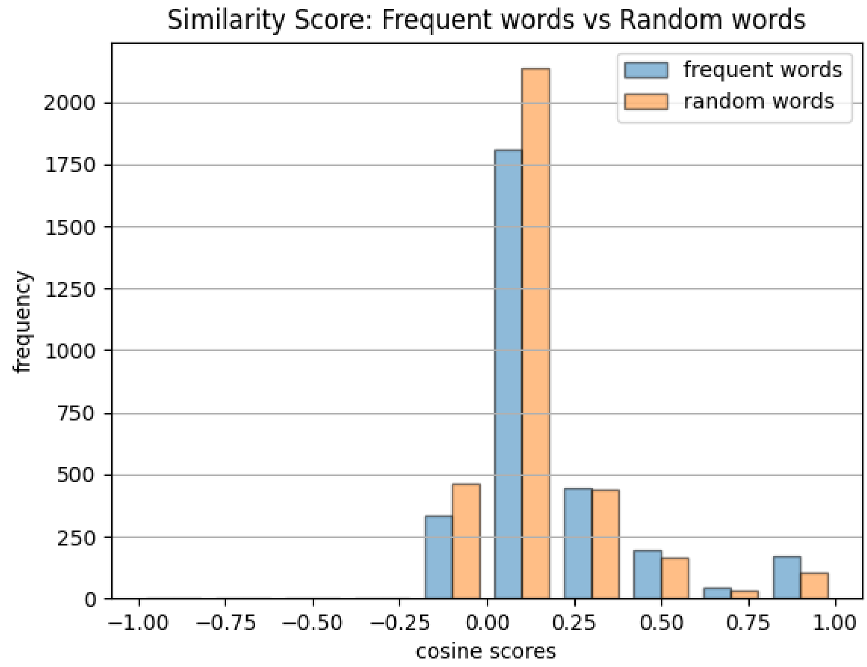


Fig. 1. Result on 500 articles, and 25 words

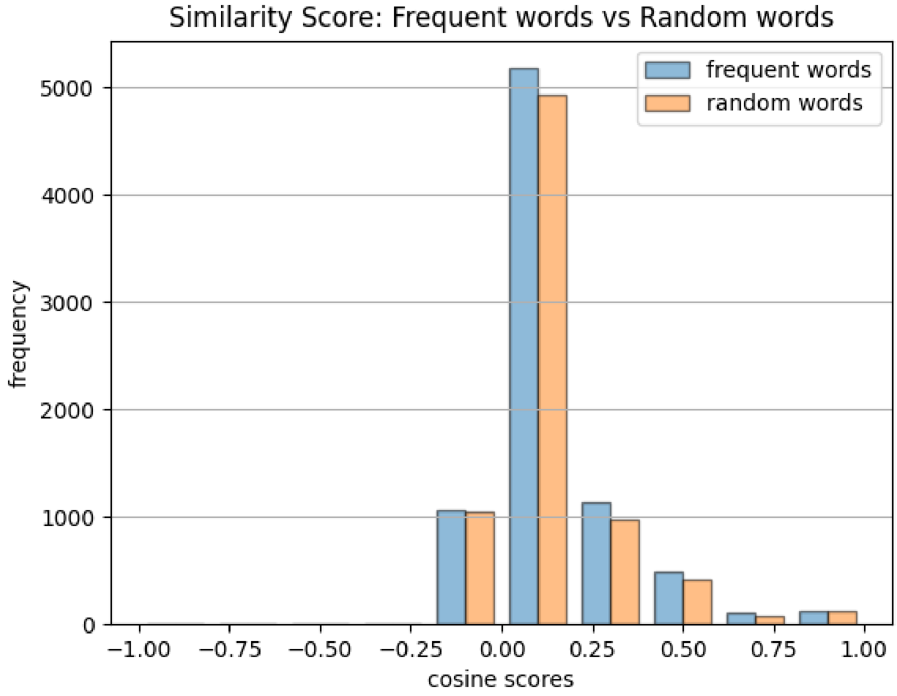


Fig. 2. Result on 500 articles, and 100 words

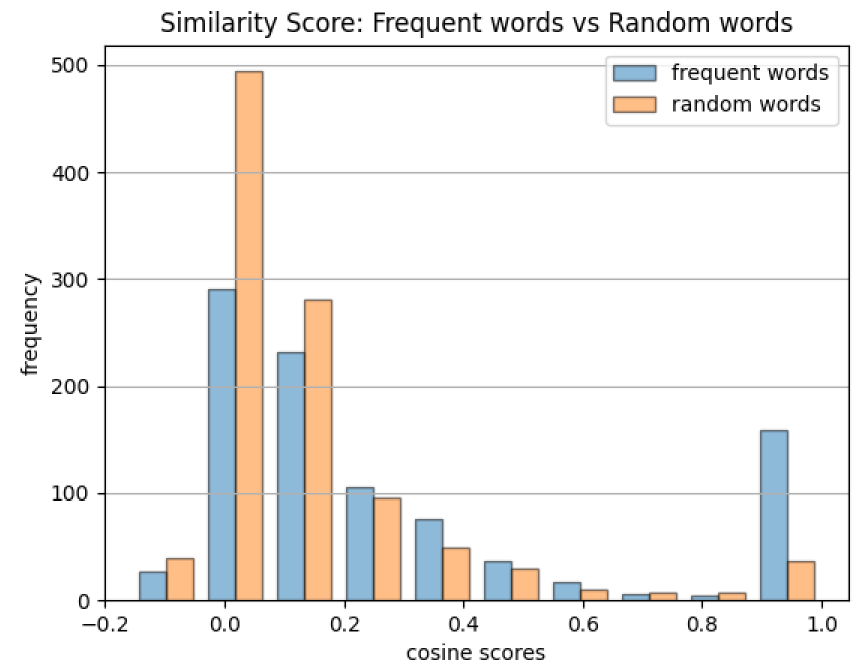


Fig. 3. Result on 500 articles, and 5 words

Identifying Trends

We can see a trend across all the figures that as fewer words are chosen, the cosine scores closer to 1.0 get higher in frequency. We can see that the frequent words are more similar to the title than the random words because in all figures, they are higher in frequency. It is important to note that the results from the three figures are made from different subsets of the data. Thus we can make a claim that no matter what article is chosen, as long as we get the top fewer frequent words, they will always have a similarity score closer to the title.

Another trend to notice is that a bigger average is concealing the difference. This is very noticeable in Figure 2 where there is almost no difference between the frequent words and the random words. This is likely because as I keep increasing the number of words, I get closer to randomness. This also explains why the histograms in Figure 2 look almost identical. The top most frequent words are the ones that are more closely related to the title. At times, they are the same words that appear in the titles. So only picking a smaller number of frequent words would make for a higher similarity score.

Discussion

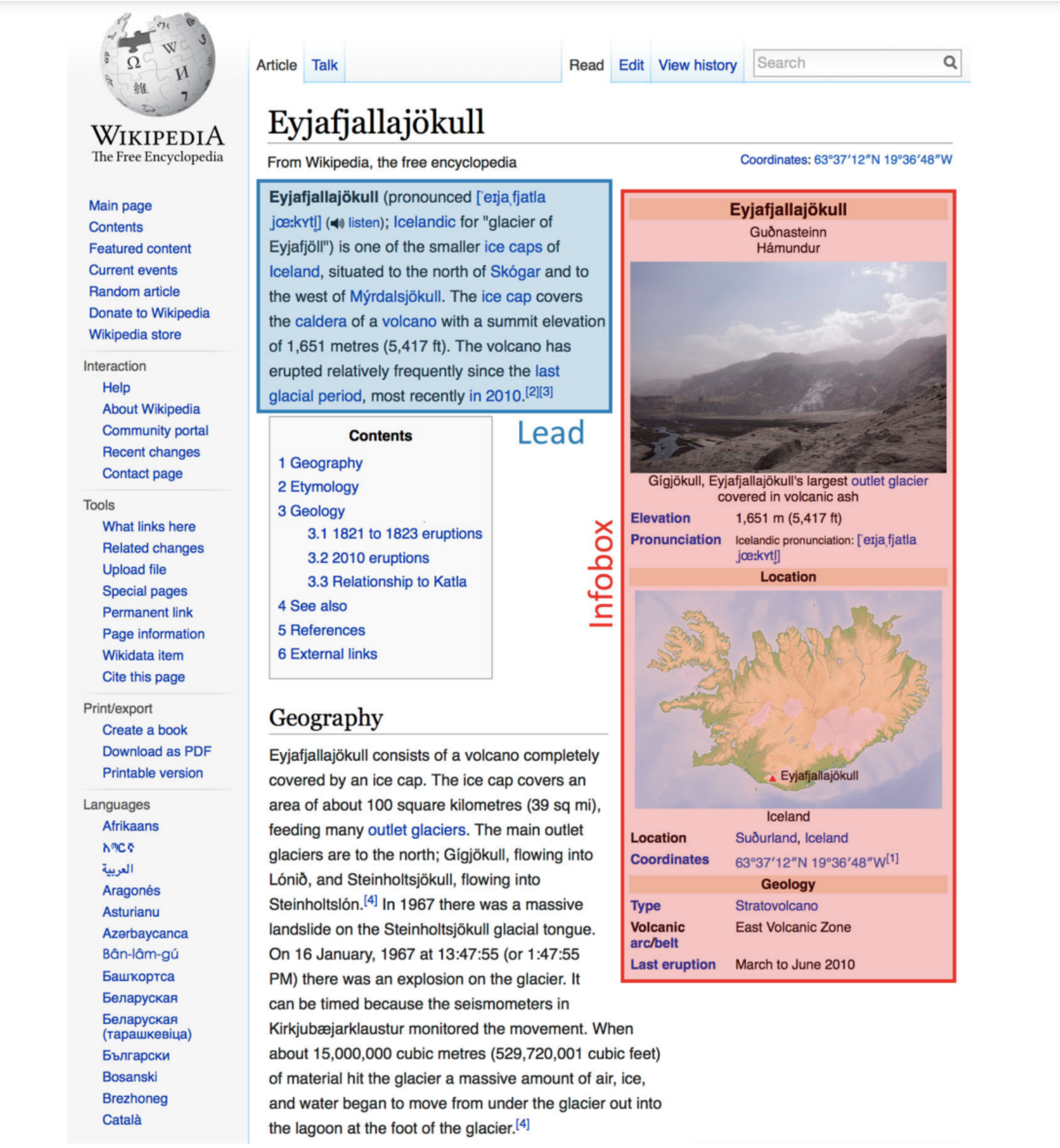
The result of the research proves that only the top few frequent words of any Wikipedia article are the most relevant to the title. My research question was how to identify keywords for content writing. Content writing is the process of writing and publishing content digitally and includes infographics, news articles, listicles, etc. In my results, I have identified that because there is a close relation between frequent words and titles, then frequent words can be used as keywords for ranking when incorporated with the proper optimization techniques.

Optimize Your Content Using Wikipedia Article Structure

My research uses the Wikipedia API, which is beneficial because Wikipedia articles always rank higher than other websites. This is because the content of Wikipedia articles are well optimized and they structure their content in ways that helps searchers easily navigate within the website. Wikipedia articles fall under the informational query classification because the articles help searchers find information about a topic. Ways in which Wikipedia optimizes its content is through its link structure and through its article structure.

Wikipedia optimizes its link structure. “Internal links in Wikipedia are typically based on words naturally occurring in a page and link to another “relevant” Wikipedia page” (Kamps & Koolen, n.d.). Wikipedia articles place links of articles that are relevant to the current page. A good relationship between each page means that they are strongly connected to your keyword or topic. It is important to build a strong relationship between pages in an article because it helps the search engine identify what page and helps the searcher easily understand your content. “The structure of Wikipedia articles, which places links to more general concepts near the top, supports navigation by allowing users to quickly find the better-connected articles that facilitate navigation” (Kamps & Koolen, n.d.). When writing content and creating multiple pages, it is important to have the links at the top navigation bar so that users can see it and be inclined to click to learn more about your keyword.

Wikipedia articles optimize its content by following a specific article structure. The researchers (Lamprecht et al., 2016) says pages are scanned in an f-shaped pattern by the users. Wikipedia articles simply follows the common website structure (Lamprecht et al., 2016). The first paragraph of a wikipedia article introduces the content and the infobox summarizes the main points to discuss. The study explored if article structure affects navigation and found that it did. The study found that users are biased and chose to navigate with links that are found in the lead section or the infobox (Lamprecht et al., 2016). Thus for writing informational content, I recommend writers to follow this format for optimization of having a lead content that introduces the topic and an infobox that lays the main topics. Furthermore, I recommend the writer to have headings and subheadings that link to the table of content.



Eyjafjallajökull

From Wikipedia, the free encyclopedia

Coordinates: 63°37′12″N 19°36′48″W

Eyjafjallajökull (pronounced [ˈeɪjaˌfjatlaˌjœːkʏtl̥] (listen); Icelandic for "glacier of Eyjafjöll") is one of the smaller ice caps of Iceland, situated to the north of Skógar and to the west of Mýrdalsjökull. The ice cap covers the caldera of a volcano with a summit elevation of 1,651 metres (5,417 ft). The volcano has erupted relatively frequently since the last glacial period, most recently in 2010.[2][3]

Contents

Eyjafjallajökull
Guðnasteinn
Hámundur

Gígjökull, Eyjafjallajökull's largest outlet glacier covered in volcanic ash

Elevation	1,651 m (5,417 ft)
Pronunciation	Icelandic pronunciation: [ˈeɪjaˌfjatlaˌjœːkʏtl̥]
Location	
	Eyjafjallajökull Iceland
Location	Suðurland, Iceland
Coordinates	63°37′12″N 19°36′48″W[1]
Geology	
Type	Stratovolcano
Volcanic arc/belt	East Volcanic Zone
Last eruption	March to June 2010

Geography

Eyjafjallajökull consists of a volcano completely covered by an ice cap. The ice cap covers an area of about 100 square kilometres (39 sq mi), feeding many outlet glaciers. The main outlet glaciers are to the north; Gígjökull, flowing into Lónið, and Steinholtsjökull, flowing into Steinholtslón.[4] In 1967 there was a massive landslide on the Steinholtsjökull glacial tongue. On 16 January, 1967 at 13:47:55 (or 1:47:55 PM) there was an explosion on the glacier. It can be timed because the seismometers in Kirkjubæjarklaustur monitored the movement. When about 15,000,000 cubic metres (529,720,001 cubic feet) of material hit the glacier a massive amount of air, ice, and water began to move from under the glacier out into the lagoon at the foot of the glacier.[4]

Figure 1. *Example of the structure of a Wikipedia article.* The image shows a Wikipedia article with the lead section (blue shading) and the infobox (red shading), followed by the table of contents and the start of the main content. In this paper we show that users focus their attention on the lead sections and the infoboxes.

Figure 1 explaining Wikipedia article from (Lamprecht et al., 2016)

How to Properly Optimize Your Content for Informational Queries

1. Identify your keyword(s) or topic. Pick 5 or less.
2. Write a leading section introducing your topic. Optional: Include links to other pages in your website.
3. Write an infobox summarizing your topic. Include pictures and links to other pages in your website if needed.
4. Write a table of contents of the headings and subheadings.
5. Incorporate your keywords or synonyms frequently within the page. Preferably after a new heading or subheadings and it the leading section, infobox and concluding section.

Tip

When creating your content, make sure to always tie it back to your keyword and to the informational need of the searcher.

Conclusion

The findings of this research suggest that given any page, there needs to be a good relationship between the top frequent words of an article and the title. The title of the page tells you what the page is about and it is also the first introduction that a visitor has of the website. When writing good content, it is important to write only the relevant information pertaining to your keyword and your search intent. As defined earlier there are three main types of search intents: Navigational, Transactional, and Informational.

A writer can write relevant information by focusing on one search intent. The benefit of using keywords with intent-specific terms when writing content is to improve your chances of being viewed by your target audience who share the same search intent. The findings of this research also suggest that if writers don't write keyword specific content, the relationship between the content and the title will be close to random because there is no keyword that tells the reader what the page is about. This finding further supports the fact that choosing only 5 or fewer keywords as illustrated in Figure 3, and writing high quality content using those keywords will create a stronger relationship between the content and the title which will help reach your target audience.

Every content writer aims to create high quality content that will help them attract their target audience. The way to attract your target audience is by creating relevant content. Although it is impossible to know what the audience is thinking when they search the internet, writers may concentrate on optimizing content, which will attract the proper audience.

Bibliography

Chotikitpat, Kittisak, et al. *Techniques for Improving Website Rankings with Search Engine Optimization (SEO)*, Advanced Science Letters, 2015, https://www.researchgate.net/publication/297661581_Techniques_for_Improving_Website_Rankings_with_Search_Engine_Optimization_SEO

Goldman, *Search Engine Bias and the Demise of Search Engine Utopianism*, Yale Law School Legal Scholarship Repository, 2008, https://link.springer.com/chapter/10.1007/978-3-540-75829-7_8

Lyons and Havig, *Transparency in a Human-Machine Context: Approaches for Fostering Shared Awareness/Intent*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2014, https://link.springer.com/chapter/10.1007/978-3-319-07458-0_18

Jansen, B. J., Booth, D. L., & Spink, A. (2008). Determining the informational, navigational, and transactional intent of Web queries. Information Processing & Management, 44(3), 1251–1266. https://doi.org/10.1016/j.ipm.2007.07.015

Das et al, Proceedings of the 19th International Conference on Management of Data, 2013, https://cse.iitkgp.ac.in/~chitta/pubs/comad13-queryGoal.pdf

Nagpal & Petersen, Journal of Retailing, 2020, https://www.sciencedirect.com/science/article/pii/S0022435920300944

YEC, *Google Ranking Factors: What's The Secret Sauce?*, Forbes, 2021 https://www.forbes.com/sites/theyec/2021/02/09/google-ranking-factors-whats-the-secret-sauce/?sh=13e1599227b6

Wikipedia Contributors. (2022, April 26). Section 230. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Section_230

Bob Johnson. (2020). The Scientific Evidence That Artificial Intelligence (AI) Will Continue to Fail, Until We Deploy “Intent.” Sociology Study, 10(2). https://doi.org/10.17265/2159-5526/2020.02.006

Chao Liu, Ryen W. White, & Susan Dumais (2010). Understanding web browsing behaviors through Weibull analysis of dwell time | Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval. Retrieved March 11, 2022, from ACM Conferences website: https://dl.acm.org/doi/abs/10.1145/1835449.1835513

Manning , C. D., Raghavan P, & Schütze, H. (2009). Evaluation in information retrieval. In P. Raghavan (Ed.), An Introduction to Information Retrieval (pp. 151–177). essay, Cambridge University Press Cambridge, England. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

Lamprecht, D., Lerman, K., Helic, D., & Strohmaier, M. (2016). How the structure of Wikipedia articles influences user navigation. New Review of Hypermedia and Multimedia, 23(1), 29–50. https://doi.org/10.1080/13614568.2016.1179798

Kamps, J., & Koolen, M. (n.d.). Is Wikipedia Link Structure Different? http://www.marijnkoolen.com/publications/2009/kamp:is09.pdf

Appendix

https://github.com/YaboDetchou/cosine-similarity-sporaj.git.

The modules and libraries that I worked with include wikipedia/bs4 for my dataset, nltk for word tokenization, gensim for similarity method to calculate cosine similarity, and matplotlib/numpy/statistics to plot. Other libraries include collections to count the frequency of words on the page, python's built in random function to get random words. The itertools.product was used in the get_similarity_score() function to find the cartesian product of each word in my title and each word in the frequent words list and random words list. So that I can loop through each word in my title (keys) with each of the values and get a similar score, if the value is present in my vocabulary. I created a class CosineSim and within are functions. I also created an empty constructor because I am not taking any input from the user. I create an object to call the histogram_plot() function to display the plot. Furthermore, I have included the page_ids of the articles I used to plot the figures.