Bookmark Classification using Multinomial Naive Bayes Model

Parth Parikh

August 2019

A browser-based bookmark serves a single purpose of storing the information for later retrieval. They are primarily composed of the following data: Uniform Resource Identifier (URI), favicon, title and time-stamp. As the number of bookmarks a user has increases, the ability to quickly retrieve their favorites can decrease significantly. Through this article, we will solve the above problem by automatically classifying the bookmarks into various categories.

Text classification is a method to map a document to one or more categories. In our case the document would be created using the bookmark data collected by the web-browser. For the time being, I will be using the following generic categories: News, Business, Programming, Shopping, Social and Technology. Each category will have relevant documents which will act as a corpus for that category. In the later part of this article I will discuss the possibility of adding user requested categories.

We start by introducing a supervised learning algorithm called *multinomial Naive Bayes*, a probabilistic learning method. The goal is to find the category which closely resembles the document to be classified. The probability of a document d being in category c is computed as

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq N_d} P(t_k|c) \tag{1}$$

where $P(t_k|c)$ is the conditional probability of t_k occurring in a document of category c, where t_k is a term which is part of the vocabulary of the document to be classified. It tells us how well would the term t_k fit in the category c. $P(c)$ is the prior probability of a document occurring in category c [1].

In the above equation, to prevent floating point underflow we add logarithms of probabilities instead of multiplying them. The revised equation would then be

$$\log P(c|d) \propto \log P(c) + \sum_{1 \leq k \leq N_d} \log P(t_k|c) \tag{2}$$

In the equation (2), the category c which maximizes the value $\log P(c|d)$ will be the optimal category for the document d. We define $P(c)$ as

$$\frac{N_c}{N} \tag{3}$$

where N_c is the number of corpuses we have for category c and N is the total number of corpuses of all the categories.

Lastly, the estimate $P(t_k|c)$ will be

$$\frac{n_k + 1}{n + |Vocabulary|} \tag{4}$$

where n is the total number of word positions in the category c's corpuses, n_k is the number of times term t_k is found among these n word positions and $|Vocabulary|$ is the total number of distinct words in all the corpuses [4].

We now define the document d, which is the document to be classified. The goal is to curate d such that it faithfully represents the bookmark's content. The first option we have is to let the website furnish us with this information while the next option would be to scrape the website's content. The former technique works well with popular websites while latter helps in categorizing uncommon sites. Most of the popular websites on the Internet provide `<meta>` tags with attributes like `keywords` and `description`. After exploring various combinations, I found that the above mentioned `<meta>` tags combined with the site's title forms the perfect document for categorizing any bookmark.

If the `<meta>` tag is not provided by the website, the site's content can be used for the document's vocabulary. In the worst case, the site may render the content dynamically. To overcome such situations, we can either use the site's URI or title to query a 3^{rd} party API which specializes in providing domain information. I have observed that Wikipedia's MediaWiki API [2] provides optimal results for semi-famous domains.

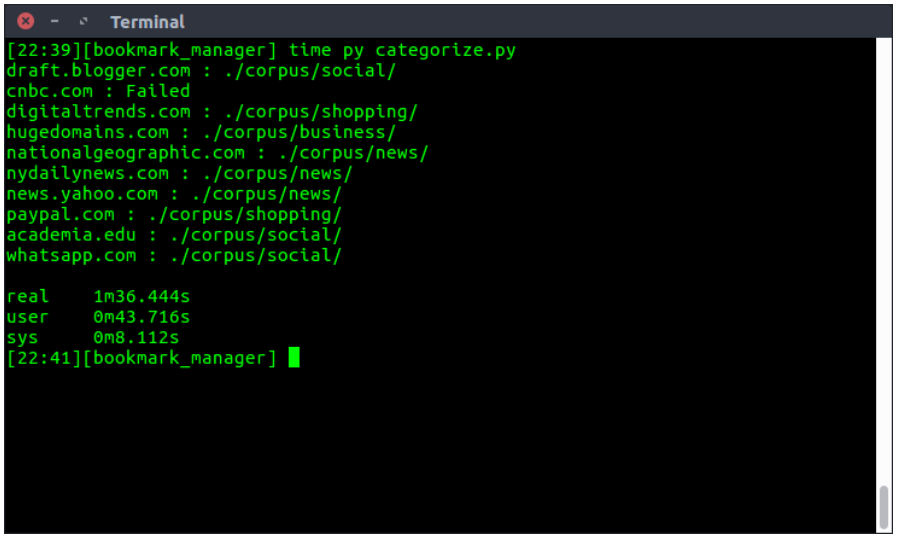

Figure 1: Ten random bookmarks being categorized using multinomial Naive Bayes model with the site's content being excluded from the document's vocabulary. Domains which encountered network error are labeled as *Failed*.

Once a document is curated, performing text normalization such as stop word and punctuation filtering, single case conversion and lemmatization can help improve the accuracy of the classification algorithm.

Another interesting approach to curate a vocabulary d as mentioned in [5], is to use the user's recent search-engine history. The keywords a user queried in the search-engine can be used in the vocabulary. This is possible because web-browsers store the time-stamp of each bookmark as it's meta-data. This time-stamp along with the URI can be compared in the user's history log to find the relevant search-engine query, which is most commonly a GET request. An example for the same is mentioned in Figure (2).

Corpuses are another integral part of this implementation. The straightforward way to design a corpus is by populating it with jargons. This method alone won't give optimal results as based on equations (1) and (4), $P(c|d) \propto n_k$. Hence the number of times the term n_k is found in corpus forms a deciding factor. By meaninglessly populating the corpus with jargons, it becomes difficult to get a good estimation for n_k. To correct this problem, we use text from various blogs, glossaries and encyclopedias. For an optimal classification, our aim should be to find a good balance between n_k and number of unique words.

Figure 2: Keywords extracted from a Google search query

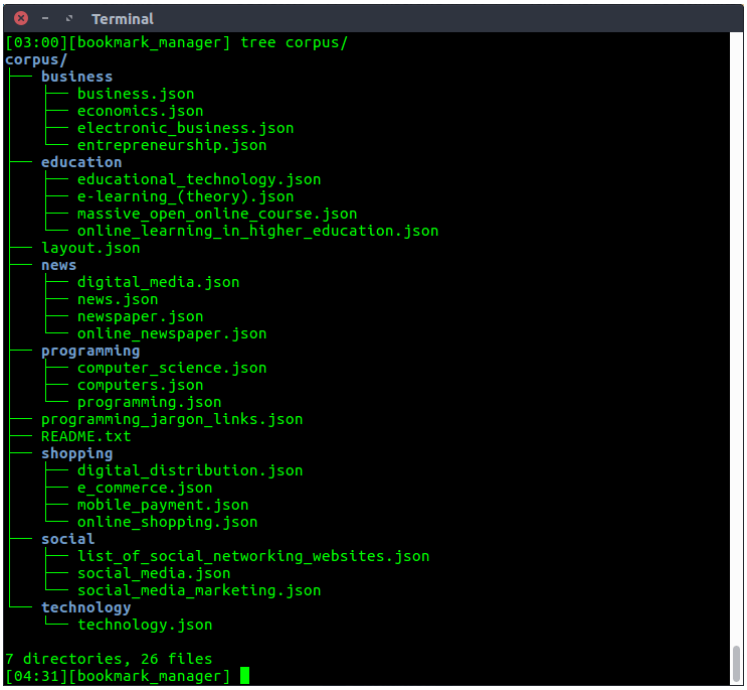

Figure 3: Generic corpuses curated for various categories

Based on my observation, corpuses curated from the categories' Wikipedia page are able to provide optimal results. One category in which Wikipedia's corpus performed below expectation was *Programming*. To improve the performance, I curated my *Programming* corpus from [3]. Figure (3) shows the list of corpuses I used during my implementation. Except for *programming* corpus, the rest are curated from Wikipedia, where the file-name corresponds to the title of their respective Wikipedia page.

Classifying user requested categories using the above technique can easily be done by curating the corpus from the relevant Wikipedia page. We can set a depth value and recursively crawl the page's *See also* section to auto-generate the corpus.

Finally, as we have used a *supervised learning* model for this project, a good future problem would be to tackle this using an *unsupervised learning* model. Furthermore, adding language detection on the site's content and increasing the language-based diversity of the corpus can further improve our training model.

References

[1] Prabhakar Raghavan Christopher D. Manning and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[2] Wikipedia contributers. MediaWiki Action API opensearch. `https://www.mediawiki.org/wiki/API:Opensearch`. Accessed: 2019-08-17.

[3] Computer Hope. Computer programming terms. `https://www.computerhope.com/jargon/program.htm`. Accessed: 2019-08-18.

[4] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[5] Chris Staff and Ian Bugeja. Automatic classification of web pages into bookmark categories. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 731–732, New York, NY, USA, 2007. ACM.