Exploring Insights from Futurice Blogs Data

CS-C3250 - DATA SCIENCE PROJECT 2022

Duong Le , Georgy Ananov , Guting Huang , Laura Talvio , Mael Chauvet , Rajat Kaul
Aalto University, 02150 Espoo, Finland
in collaboration with Futurice
supervised by Jorma Laaksonen, Selen Pehlivan, Rachhek Shrestha and Seth Peters
{duong.h.le | georgy.ananov | guting.huang | laura.talvio | mael.chauvet | rajat.kaul}@aalto.fi

***Abstract*— As a means of bolstering sales, marketing, and talent acquisition efforts, Futurice frequently publishes blogs created by the company's employees. Blogs serve as one of the company's public outreach channels and thus constitute an integral part of Futurice's brand, as well as play a pivotal role in attracting targeted clients and talent, establishing and maintaining business relationships with customers and other companies. However, writing timely, engaging blogs is challenging, even with the help of existing practices, such as search engine optimization. In this project we analyzed past blogs from Futurice to provide an overview of their history and context and to leverage gathered information for a simple recommendation system.**

As an output of the project, we have constructed an interactive web page that showcases results obtained from the conducted analysis, offers an online blog editing assistant, and houses a rich visualization tool where users can explore relevant statistics about past blogs.

I. Introduction

This project was prepared for the course CS-C3250 Data Science Project 2022 at Aalto University in collaboration with Rachhek Shrestha and Seth Peters at Futurice.

Blog (a truncation of "web log") is commonly defined as a frequent online publication, detailing personal thoughts, ideas, and experiences. In recent decades, blogs have found wide popularity as a means for companies to forge and support relations with customers, share information about the company's offerings and promotions, as well as to attract the attention of potential employees. Such publications, commonly referred to as "corporate blogs", add an informal conversational aspect to otherwise formal corporate public relations, fostering a more personal connection between the business and the customer. At Futurice, blogs are created by a wide range of employees, with the direction for topic selection being obtained through Search Engine Optimisation (SEO) techniques and insights from the sales team.

While blogs can be a highly effective tool in the public relations arsenal of Futurice, the resource costs of frequently producing high volumes of blog articles can be significant. The high resource cost necessitates designing blogs in a way that generates high engagement for the published material. The strategies that are currently in place primarily focus on the selection of engaging topics for the blogs, and are informed to a significant degree by external observations (such as SEO). In order to better cater the blog design to the specific audience of Futurice clients, it would be reasonable to incorporate internal observations into the blog-writing strategy. To that extent, we have decided to turn to records of the popularity of existing Futurice blogs in an attempt to identify trends that are specific to the behaviors of the audience of Futurice blogs.

Our goals for this project can be summarized as follow:

- Create a tool that helps writers to understand the structure of the existing blogs archive at a glance
- Find out which aspects of a blog contribute to its success
- Develop a system that can provide suggestions on how to optimize the blog based on observations about previous successful blogs

To meet these goals, it is important that we can identify the correct research questions. When progressing with the project, these are the questions addressed in our project:

RQ1: Do the current topics represent the actual structure of the blogs? Currently, there are approximately 800 blogs and 12 different categories in total. Some of the categories are quite similar to each other (e.g., Emerging Tech and Innovation & Design), while some other categories are ambiguous (e.g., Ways of Working, Learning). For that reason, we want to see if we can cluster the blogs using less number of clusters compared to the current categorizing.

RQ2: What are the most popular keywords in different periods of time? This is one of the questions directly asked by the representatives of the company. By visualizing this popularity, it can help to see how the writers in Futurice reacted to some major events in the world.

RQ3: What features of the blog affect its success? Blogs can have a wide range of features, such as text length, the number of difficult words used in the blogs, or the voice of the writers. But it can be difficult to optimize all of those features when writing a new blog. Thus, we want to extract and find out which features contribute the most to the success of a blog. The success here can be measured using different metrics, such as pageviews, or the average time that viewers spent reading the text.

RQ4: How to make small decisions when writing new blogs? It is difficult to make a personalized tool to optimize minor decisions when writing a blog. For this reason, we want to develop a general visualization tool that can visualize the statistics of the blogs, and how these statistics change through time. This tool can help us deliver the data to the writers in a more concise and informative way.

II. Data

The project used data from three different sources: blog data from Futurice's webpage, Google Analytics data collected by Google for Futurice, and Google trends data, from which the main emphasis was on the first two. In the following, we will go through how these were collected and then preprocessed to be combined and used for analysis.

A. Data Collection

1) Scraping: Our journey to scrape the data can be broadly classified into 3 stages:

- Text pattern matching
- HTML parsing
- DOM parsing

The first iteration of the scraper downloaded the web page by sending an HTTP GET request using Requests, which is a Python library that allows the user to send HTTP requests very easily. It then used regular expressions (hereon: regex) to search through the blog text. This works by pattern matching a parameter string, which contains the search parameters, with an input string. The input string, in this case, is the entire output of the HTTP GET request i.e. the HTML of the web page.

This was successful to some extent, but could not be relied upon for good and clean data, since regex can not handle variations between web pages. This lead to random punctuation marks and characters scattered throughout the text, which added noise to any models that we trained on it. Still, it was very useful as a minimum viable product, and it allowed the team members who were researching text processing and clustering to get prototyping done and obtain preliminary results.

The second iteration of the scraper still used Requests to download the HTML contents of the web page, but it then fed the downloaded HTML into BeautifulSoup, which is a Python library that creates a parse tree from the contents of the web page. This now allowed us to extract the contents of HTML elements of the web page cleanly. Also, since we now extracted by HTML tag, the scraper had some tolerance for variation between articles. This still was not an ideal solution, since if an article was formatted in a unique way, the data would end up going to the wrong location in our database, or even worse, the HTML element would not exist for that article resulting in an error.

In all versions of the scraper, a database was created by assigning features like Article Name, Date of Publishing to variables in a Python object which represented the blog, and converting a list of those objects into a Pandas DataFrame, which is a useful way of tabulating data in the python data analysis library Pandas.

The final iteration of the scraper addressed a key issue, which is that modern web pages heavily rely on JavaScript to load them in their entirety. Certain portions of the article failed to appear in the previous scrapers, because Requests download only the first output of the HTTP GET request, and doesn't wait for the JavaScript elements to load. This was solved by changing from Requests to Selenium, which is a Python library that supports browser automation. Using Selenium, we could now download the blog as a reader of the blog would see it. Then we used Selenium's inbuilt parser to find the relevant elements in the Document Object Model (DOM), which is similar to a parse tree, and convert their contents to text and add them to the blog object.

2) Google Analytics data: We received the Google Analytics data directly from Futurice in the form of three separate Excel tables. Each record within the tables contained five variables:

- URL of the blog in question
- number of times the blog has been viewed
- average time spend on the page
- bounce rate for the included visits
- exit percent for the included visits

NOTE: To preserve the confidentiality of the provided data, all information related to Google Analytics has been replaced with randomly generated noise for the public version of the project. All conclusions drawn from the provided data have also been redacted.

3) Google Trends data: Google Trends data is data about search word popularity that Google gathers from its users. This data was accessed by using an unofficial python API pytrends [21]. As input, the user gives a time window, up to five search words, and a geolocation. Google returns a data frame containing dates and corresponding scores, where a single date-search word pair that has the biggest score is scaled to 100 and everything else is scaled to that. This makes the data rather ambiguous to work with. When taking into account that google limits the number of queries one can make, working with Trends data gets rather challenging.

B. Data Pre-processing

1) Scraped Blog Data: As we access the data from within the elements in the DOM, there is very little pre-processing that needs to be done. In the case of certain blogs (those that use tables or embedded elements) further pre-processing could be done to clean the data (of escape characters, special characters like '—'), but currently, they are left unchanged.

2) Google Analytics Data: Before the Google Analytics data could be used, it needed to be aggregated in a way where each blog is only included in a single data entry that would contain all available information about the analytics for that blog. We carried out the aggregation with tools built into the `pandas` framework. We then matched up the resulting clean analytics data with the blog data collected through scraping.

3) Google Trends Data: Google Trends data did not require preprocessing, as Google preprocesses the data itself too much already to be ambiguous and only comparable to itself.

III. Feature Extraction

In addition to the data provided by Futurice, we aimed to extract more features from the data to analyze which features contribute to the success of the blog posts. The features we were able to extract were average sentence length, text length, readability scores, sentimental scores, and lift score. This section discusses the methods used to extract these features.

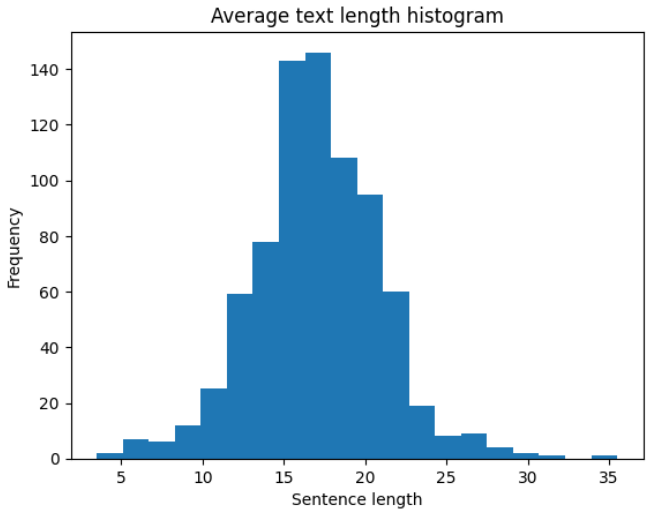

Figure 1: The distribution of the average text length

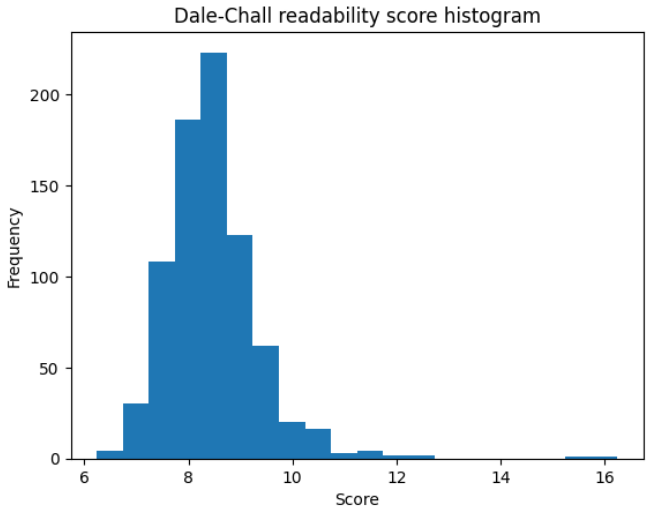

Figure 3: The distribution of the Dale-Chall readability score

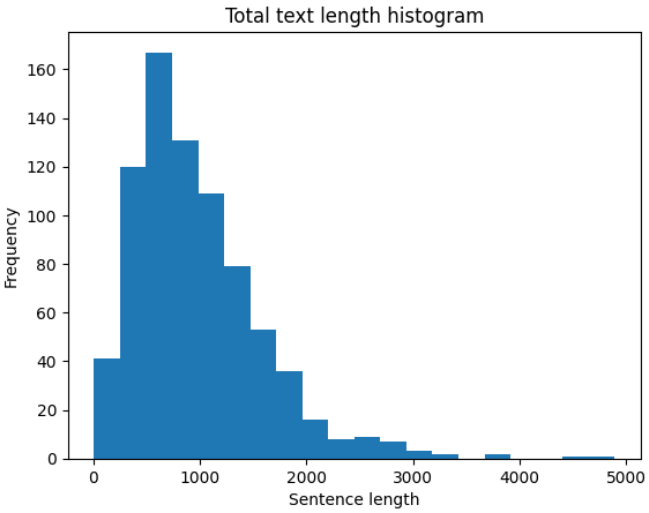

Figure 2: The distribution of the text length

A. *Average sentence length*

First, we used NLTK [6] to tokenize the blogs into multiple sentences. Then we used regex to filter sentences that have the form "1." as these sentences are only used to signify a chapter or section. We also used regex to filter newline characters. Finally, we collected all of the filtered sentences and calculated their mean length. Figure 1 shows the average text length distribution of the blogs. It can be seen that most of the blogs have around 17-20 words per sentence.

B. *Text length*

Most of the calculations for this statistic were exactly the same as those for the average sentence length. The only difference is that at the end, we calculated the sum of the sentence lengths instead of taking the mean. The distribution of the text length can be seen in Figure 2. The histogram is skewed to the right, with most of the texts having around 1000 words.

C. *Readability score*

For the readability score, we calculated the Dale-Chall [13] and the Flesch [14] scores. The Dale-Chall readability score takes into account the sentence length and the number of difficult words for fourth-grade American students to comprehend. The score is ranged from 0 to over 10, and the higher the score, the harder it is to read. The Flesch score, on the other hand, considers the total number of words and sentences in a text, along with the total number of syllables in that text. The intuition behind the score is that the more syllables the words have, the harder it is for the reader to understand. The range of the Flesch reading ease score is from 0 to 100, and in contrast to Dale-Chall scores, lower scores indicate that the text is harder to read. The reason why we choose these two scoring systems was that they cover different aspects of the text, and it would be better if the users have more options to gain insight from the text.

The distributions for the Dale-Chall and Flesch scores can be seen in Figures 3 and 4, respectively. It can be seen that both scores agree that most of the texts are suitable for $11^{th} - 12^{th}$ grade.

D. *Sentimental scores*

The sentimental score algorithm [19] in NLTK library gives four different scores. Positive, neutral, and negative scores give a percentage of the words in the input text that are labeled to be of the corresponding sentimentality. The sum of these scores is always 1.00. In addition to these, there's also a compound score, that tells if the text was in general classified to be a positive (closer to 1.0), negative (closer to -1.0), or neutral (closer to 0.0). This implementation works only for English texts.

Currently, Futurice's blogs consist on average of 82% neutral words, 15% positive words, and $<3\%$ of negative words as can be seen in Figure 5. The sentimental scores indicated that some minor tweaks in them could be related to more successful blog posts. Still, in general, they did not show great

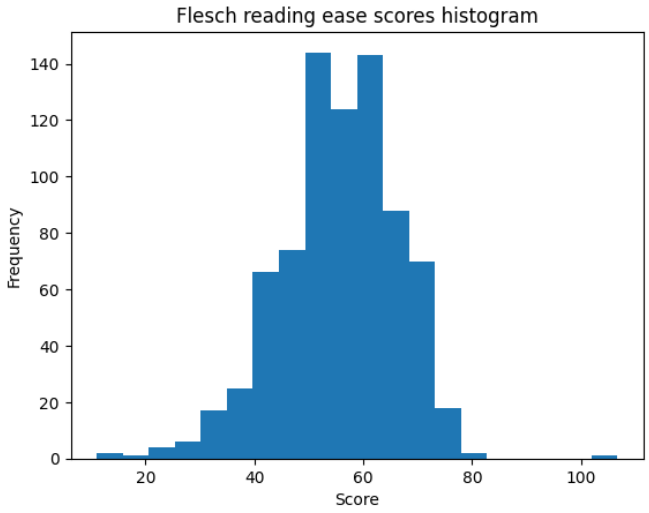

Figure 4: The distribution of the Flesch reading ease score

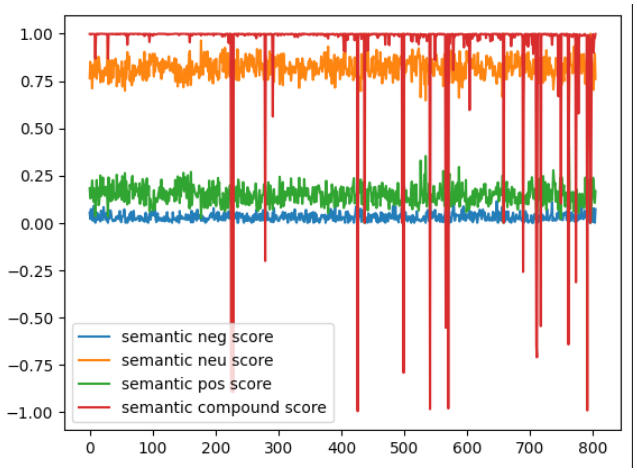

Figure 5: The percentages of different words with specific sentimental attributes in one blog text across all the blogs

relevancy, and the changes in scores were so small (a couple of percentage points) and close to the medians, that any human writer could not follow those guidelines sensibly, and thus the result of sentimental scores is not discussed.

E. Lift scores

Lift score is a metric that tries to capture if the article has been covering a topic that has been gaining more and more popularity during its publication time. For each text, five keywords were searched by using maximal marginal relevance (MMR) [22] [23]. First, words that were too similar were filtered out using cosine similarity. Then five keywords were picked so that they were coherent enough with the whole document, but were different from each other which is controlled by a parameter called diversity, which we set as 0.4. Finally, for each keyword, we calculated how much the word has been feeling lift at the time of publication by calculating the relation of the publication months scores with the average score for the past year queried from Google Trend [20] by using the pytrends library [21]. The score for the whole blog text was counted just by taking a sum from all the scores.

IV. Methods

This section presents the methods and approaches that we used to answer the research questions mentioned in Section . At the end of every subsection, we discuss the results of the corresponding experiments.

Section IV-A demonstrates the clustering methods that we used to find a way to cluster the blogs using less number of clusters. Section IV-B presents our approach to extract the latent topics of the blogs with BERTopic. Section IV-C shows the process by which we created the keyword popularity graph. In Section IV-D, we tried to assess the importance of the features of the blogs to see how well they can be used in predicting the success of the blogs. Finally, Section IV-E shows the process of making an assessment tool that can tell the users how well their new blog compared to the existing ones.

A. Clustering

The first method we decided on using to answer RQ1 was a clustering model. Through this type of model, we could not only start exploring the accuracy of the current topics Futurice chose to represent their blogs, but also get a clearer image of which types of blogs tend to perform better.

1) Initial Pre-processing: As clustering models are generic models not optimized for natural language processing, it was imperative that we cleared the features we'll be providing the models of any irrelevant information. Hence, each blog was lowercased, lemmatized, as well as cleared of contractions, punctuation, numbers, as well as stopwords obtained through the NLTK [3] library. Through this process, we were now left with text retaining only the important information of their original corresponding blog. In order to use this text in a clustering model, we transformed these texts into numerical data that could be interpreted by the model. TF-IDF is a method which measures the relevancy of a term in a document relative to a corpus, and represents these documents based on this relevancy, which made it perfect for our future clustering models to train on. Hence, we decided on using scikit-learn's TF-IDF Vectorizer [11] with `use_idf` set to True, and `norm` set to 'l2', as those two parameters seemed to give the most accurate representations for later parts of the process. Before attempting to train models on our data, we first explored our numerical data to get an intuition of what we should obtain in our clustering models later on. We began by first exploring the data visually. The numerical vectors obtained through TF-IDF are of high dimensionality, meaning we needed to reduce them to 2D vectors in order to be plotted. Scikit-learn's TSNE [10] was used to do so, with its parameter `n_components` set to 2, resulting in outputs of 2D vectors. The plotting of these 2D vectors are visible in Figure 6. In order to get a more concrete understanding of the plot, we also trained a KNeighborsClassifier [**?**] from scikit-learn, with the TF-IDF vectors as the features, and the current topics of the blogs as the labels. Plotting the classifier's confusion matrix, visible

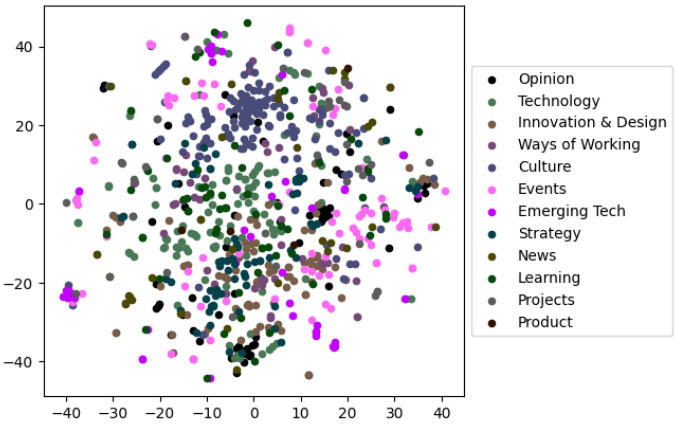

Figure 6: Blogs plotted based on their TSNE components

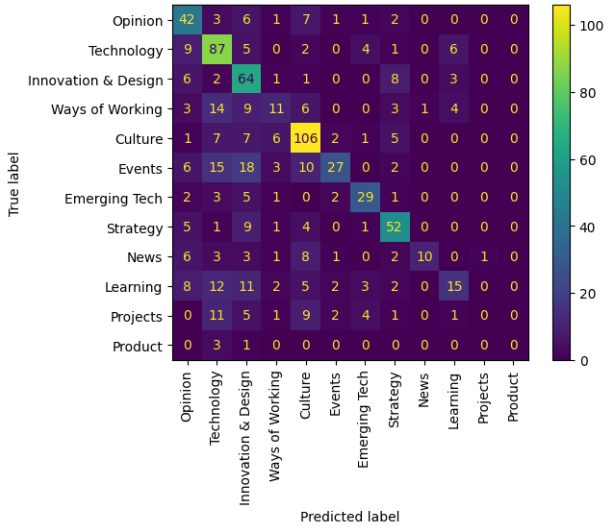

Figure 7: KNeighbors Classifier Confusion Matrix

in Figure 7, complemented our visual data with numerical data. Interpreting both, we noticed two main ideas. The current topics did not seem to properly represent the blogs. This was both noticeable visually through the constant overlap of colors throughout the graph, but also in the confusion matrix, where many topics seemed to be wrongly classified. However, we must also acknowledge that two of these topics seemed to be extremely classifiable: Culture and Technology. They were both relatively isolated on the graph, with very little overlap from other topics, but were also highly classifiable by the KNeighbors Classifier. This first analysis hence directed us towards the idea that there were better topics to classify the blogs, but that Technology and Culture should however remain as they are. The latter will serve as a good metric when choosing a clustering model.

2) Clustering Model Development: Before continuing on, let it be known that the visualizations you will see throughout this section are indeed using the same TSNE vectors as the previous one. They may look different but are just a rotation apart from the original. Let us now begin the process of choosing a clustering model with good performance. After some initial tests with different clustering models, we determined that the KMeans [8] and Agglomerative Clustering [9] models had the best performance out of them all, the two generating clear clusters with some differences. The process of then choosing the model that presented greater "performance" was done through a very brute-force method. As both these models allow you to choose the number of clusters through the parameter `n_clusters`, each model had the contents of each of its clusters manually checked and evaluated for all cluster amounts ranging from 4 to 13. While this process was lengthy and inefficient, it facilitated the exploration of the results of the clusters in great detail. This allowed us to determine that KMeans not only was more consistent with its clustering, with clusters usually revolving around similar ideas and topics even after being ran multiple times, but also more accurate, with clusters that were more logical and clear in their topics than in Agglomerative clustering.

3) Clustering Model Evaluation: With KMeans now selected as our model, we can start putting some thought into picking an 'optimal' `n_clusters`, for our model. A very simple approach to the problem would have been to run a model with a set `n_clusters` many times, get an average of its silhouette score, and choose the `n_clusters` with the highest average. However, our purpose in this clustering is not to get an 'ultimate' clustering model which can classify each blog perfectly, but instead to determine clusters that serve as better categories than the currently existing ones. Hence, in a similar way as when choosing our clustering model, we took the time to explore each `n_clusters` possible in a range from 2 to 15 and the resulting clusters from the model. From this study, despite using a KMeans algorithm, we found a pattern very similar to that of an Agglomerative clustering model. Clusters produced when selecting lower values for `n_clusters` would naturally get split into more specific clusters when we increase `n_clusters`. Hence, instead of a strict hyperparameter tuning, dictating a specific amount of clusters our model should have, we now instead obtain a range of precision(Roughly between 3 and 11 clusters), correlated to the number of clusters, from which we can pick any value depending on our objective. For our current task, `n_clusters` set to 10 was determined to be the value balancing precision and grouping best out of all values. Using this as our parameter, we get the clusters represented and named as in Figure 8. As a helpful note, the 'Company' cluster is one that contains all company-level strategy blogs, while the 'Strategy' cluster contains blogs oriented toward project strategy.

4) Results: As these clusters group blogs into topics that represent their content more accurately, we can make use of these clusters to gain insight into the performance of the blogs.

[REDACTED]

B. Topic Modeling

As an additional method to explore the structure of the blogs to answer *RQ1*, topic modeling was applied. We chose BERTopic [1] as the modeling technique, because it is easy

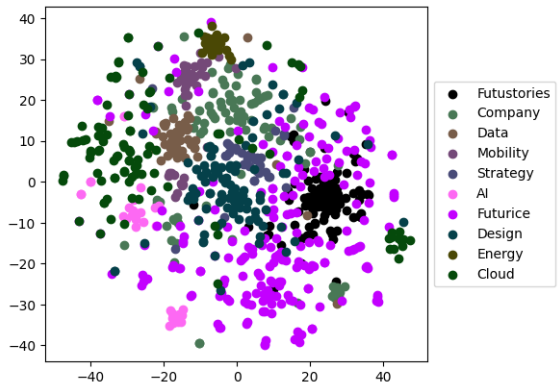

Figure 8: KMeans Clusters with ten clusters

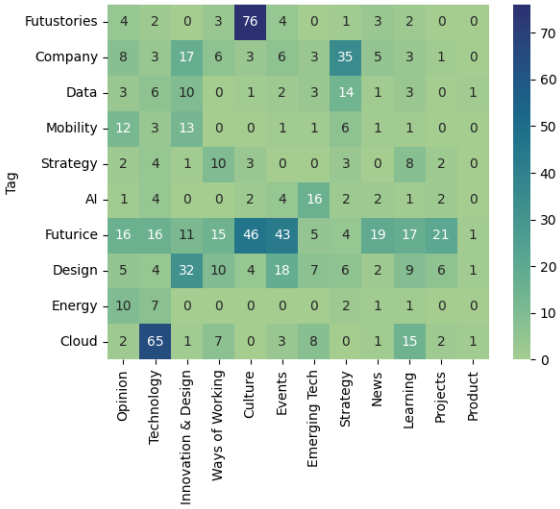

Figure 9: Heatmap representing which blog tags are in which clusters

to use, produces coherent topics, and provides a set of comprehensible, customizable visualization tools. BERTopic takes advantage of SBERT embeddings [24] and c-TF-IDF [1] to produce interpretable topics without the loss of important keywords. Different from certain techniques, BERTopic also takes into account contextual information. The benefits of using BERTopic extend from the fact that the structure of the algorithm used enables much flexibility for adjustment to fit our own use and that variations such as guided topic modeling and dynamic topic modeling are easily extendable. Furthermore, interactive visualization methods included in the toolkit makes it preferable for demonstrating results to people from various background.

1) Initial Pre-processing: While much of the hard labor was handled by BERTopic, some pre-processing needed to be performed. Each word is lowercased and lemmatized. Contractions, punctuation, numbers, stopwords, and short words (word length less than 3) were removed as they were insignificant to the meaning of a text for modeling topics. A list of stopwords from the NLTK python library was used as a default list, on top of which the word "futurice" was also removed. When exploring the blog articles, we noticed that some articles are either not written in English or contain words from another language, such as Finnish and German. These words caused noises among the fitted topics and therefore needed to be removed, using the NLTK library. NLTK's Wordlist Corpora [2] includes a collection of English words from the dictionary, which we used to match each word of our text to check whether the word is English.

2) Topic Model Development: BERTopic's algorithm is mainly composed of four parts: document embedding, document clustering, and topic representation [1]. `BERTopic`'s default dimension reduction and clustering algorithms are UMAP and HDBSCAN respectively, where each has default parameters set. Using the default values, however, produced very few topics, where most documents were classified as outliers. This was most likely due to the small number of documents we have (= 785). Reasonably large datasets were usually required for topic models to produce proper results, ideally, over 1000 [3]. In our case, it was not possible to increase the number of documents if we only focused on articles from Futurice. However, we were working with a dataset where each document had a longer sequence length, in contrast to product reviews or tweets. Thus, it was still possible to extract good results from 785 documents [4]. In the following discussion, we will go deeper into defining our own parameters for `UMAP` and `HDBSCAN` with the goal of producing a reasonable number of coherent topics.

- **UMAP** For high-dimensional embeddings, it was necessary to reduce their dimensionality before feeding them to HDBSCAN to make it easier for the clustering algorithm to find the clusters. Furthermore, UMAP was used to map the documents and their topics to 2D for visualization. The `UMAP` library was used [26].
  The `UMAP` algorithm takes in four major hyperparameters, `n_neighbors`, `min_dist`, `n_components`, and `metric` [25]. Two parameters were adjusted: `n_neighbors` and `random_state`. The stochastic nature of `UMAP` generates different outputs each time the same code is run. Hence, as Grootendorst [3] suggested, a `random_state` was set. It is also realized that some random states were preferred over others in terms of the diversity and number of topics generated. After trying out several options, we set `random_state` to 55.
  The other parameter, `n_neighbors`, specifies the number of high-dimensional neighbors each point has and plays an important role in adjusting the clusters to focus more on local or manifold structure. A typical default of `n_neighbors` is 15. Increased values enable UMAP to see a broader structure, while decreased values enable UMAP to have a more local view. Since we already used other clustering methods to assess broader categories, when topic modeling we wanted to focus also on local clusters to discover some hidden topics not present in current categories for the blogs. The model has been run on `n_neighbors` ranging from 10 to 20 and compared by the silhouette score on the resulting cluster. Among them `n_neighbors = 11` (0.5058) and `n_neighbors = 18` (0.5026) obtained the best scores. Without losing the manifold structure

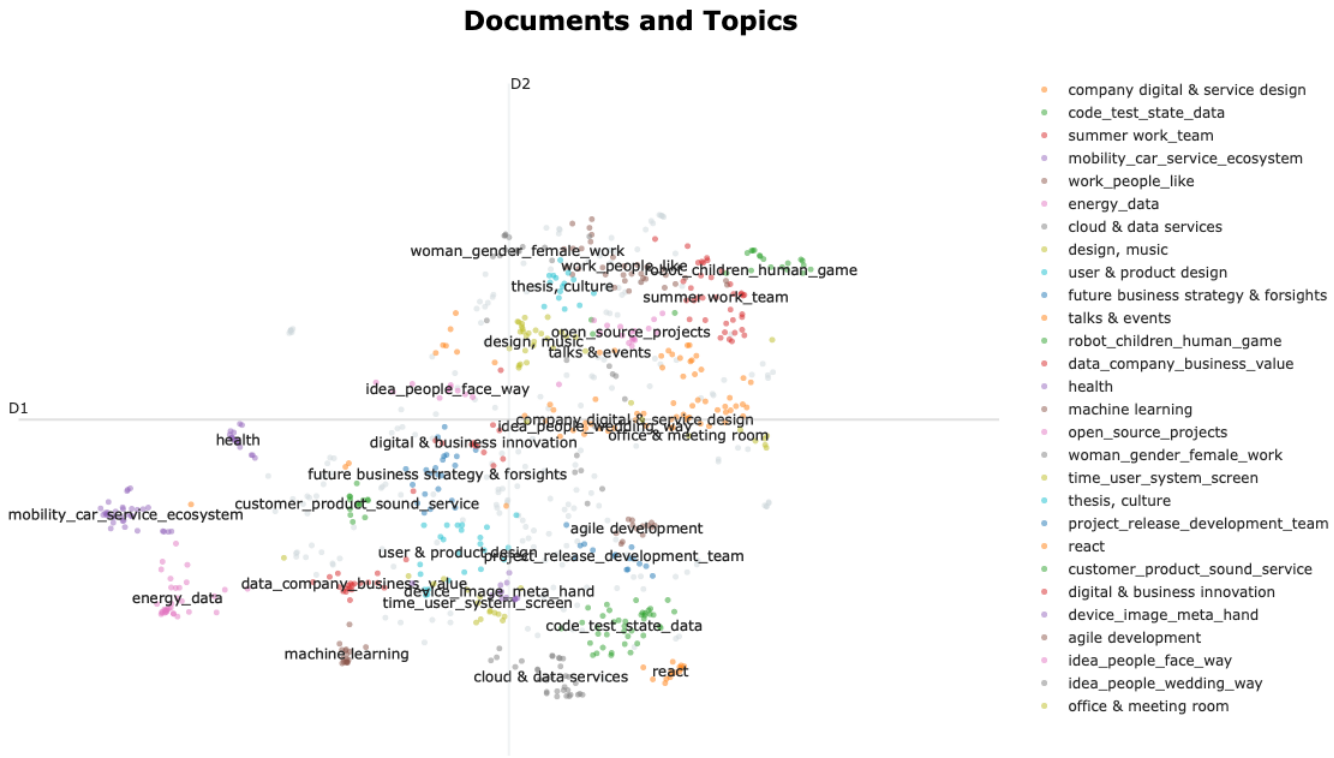

Figure 10: Document clusters with corresponding topics
Interactively on streamlit: includes also associated URLs for each data point

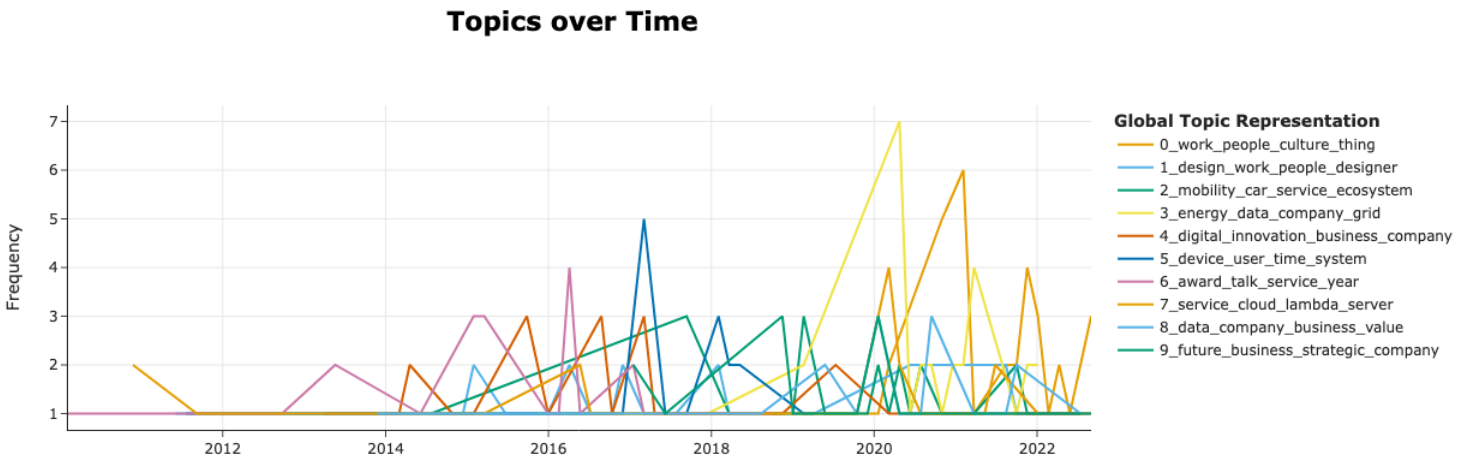

Figure 11: Change of top 10 topic's frequency over time.
Interactively on streamlit: Each topic's representative keywords over time

nor getting a large number of excessively detailed topics, `n_neighbors` is set to be 18.

- **HDBSCAN** The only parameter modified in the HDBSCAN model was `min_samples`. Initially, apart from having a very small number of topics, most of the blogs are labeled as outliers. Less noise will be generated if the parameter value is lowered. It is, however, worth noting that lowering too much Starting from the default value 5, the value is lowered and compared by the resulting BERTopic model's silhouette score. Eventually, `min_samples = 3` with a score of 0.5077 is chosen.

Before feeding the tuned models and algorithms to `BERTopic`, a `seed_topic_list` was provided. The seed topic list will be used by `BERTopic` as a base to try converging to when running the algorithm. The list consisted of topics that were known to exist in the document collection and were used as a guide for the model. In addition to the existing list of categories, "Futustories" was added. If, however, those topics do not exist or can be split into more topics, they will not be modeled by the algorithm.

In addition to the basic BERTopic modeling, a variation of it called dynamic topic modeling was also used to showcase the change of the most relevant keywords for each topic over time. The evolution of the top 10 topics can be seen in Figure 11

3) Topic Model Evaluation: The quality of a topic model is usually about the human-interpretability and diversity of the produced topics. It can be difficult to evaluate depending on the task the topic model is created for. In our case where topic modeling is used for document exploration, the outcomes were not as clearly measurable as classification tasks.

When tuning the parameters for the clustering part of the BERTopic model, we utilized the silhouette score to assess the quality of the clustering. For the outputted topics the main evaluation method used is human-judgment. The topics are visualized in tabular form and with the help of bar charts of each topic from the BERTopic package, the 28 topics examined. Furthermore, the built-in interactive visualization of the document clusters and their topics is used to check the documents with their corresponding topic (see Figure10).

4) Results: The interactive visualization of the document clusters and the topics is a useful tool to examine the topics and documents with that topic. Regions of the plot on our simple online application can be selected to zoom in on each topic cluster for a closer examination. Moving the cursor to a data point shows the corresponding URL of that document. Unfortunately, with the current implementation, the URLs are not clickable, but since the URLs contain the titles of each blog, it can still be useful to observe articles that get listed under a certain topic. For future blog writers looking for inspiration, the plot can be used, for example, to search for some articles on a certain topic.

As seen from Figures 6, 8, and 10, many of the documents have overlapping topics. From Figure 12 we observe that the top topic, which contains the most number of documents, is more similar to each of the other topics than the other topics between themselves. The topics are relabeled if they are easily interpretable and generalizable, for

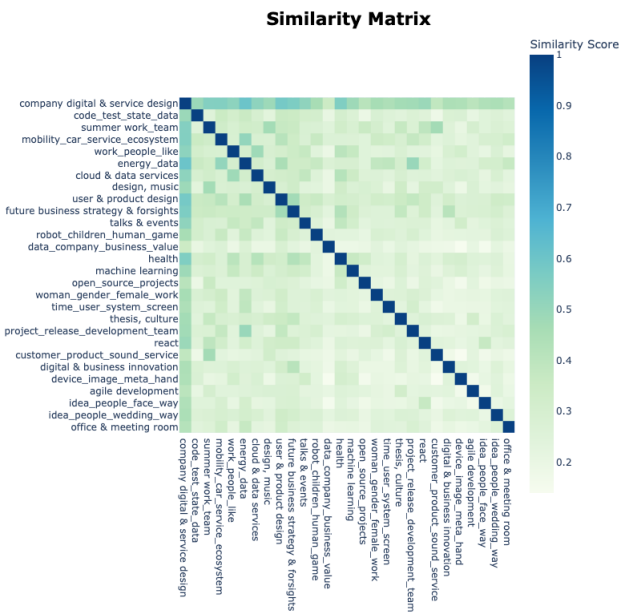

Figure 12: Similarity matrix showing the similarity scores between each topic

example, `future_business_strategic_foresight` are changed to future business strategy and foresight.

Some of the new topics that are part of the original broader categories that are revealed by the topic model include health, mobility, machine learning, children_robots_education, and energy. Among these new topics, some such as health and mobility are known to be closely related to work done at Futurice. We also noticed when examining the topics with their documents that the titles of most articles align with the topics generated when the data used for fitting the model only includes the main contents of the blogs.

C. Keywords popularity

One question that was directly asked by the researchers at Futurice is to find out which word/phrase has the highest frequency in a period of time *RQ2*. We saw that this direction is simple and informative to pursue so we developed an interactive graph to visualize this. In this application, we defined keywords as words that are not stopwords. This made the algorithm simpler as we did not have to take into account which words are important.

1) Initial Pre-processing: Before plotting the graph, we first used the `date_range` function to split the blogs into different batches corresponding to different month periods. After having the batches, we linked all of the texts in that period together. We then used `CountVectorizer` [18] to filter stopwords and extract the ngrams and their frequencies. In this project, we only filtered stopwords in the preprocessing, we did not use Lemmatize or any Language Detection models. This results in similar words such as 'Revolution' and 'Revolutionise' could be categorized as two different words and also some Finnish words were mixed in the results. We did try some Language Detection, but it immensely slowed down the

	2010-03-31	2010-05-31	2010-09-30	2010-10-31	2010-11-30	2010-12-31
10 years	0.50	0.0	0.0	0.0	0.0	0.0
150k pretty	0.25	0.0	0.0	0.0	0.0	0.0
1600 amphitheatre	0.25	0.0	0.0	0.0	0.0	0.0
2001 2008	0.25	0.0	0.0	0.0	0.0	0.0
2001 believed	0.25	0.0	0.0	0.0	0.0	0.0

5 rows × 141 columns

Figure 13: A part of the calculated frequency data frame

code, so we decided to not include it. For the Lemmatization and Stemming, the algorithm we used filtered out words like 'git', and we believe it is these words that should be taken into account. During a few test runs of the algorithm, we observed that using bigrams would lead to fewer Finnish words appearing in the graph. For that reason, we decided to use only bigrams.

2) Keyword popularity model development: Each `CountVectorizer` has the method `get_feature_name_out`. This method returns a list of the bigrams and their count. We can apply the Vectorizer to each of the linked blogs, so we can obtain the list of keywords grouped with their count. We also kept track of how many blogs were there in each of the time periods. Thus, it was straightforward to calculate the frequency by dividing the count of the words by the number of blogs. These frequencies are then collected to a dataframe to pass to the plotting algorithm. A part of the dataframe can be seen in Figure 13. Finally, to plot the graph, we use the library `raceplotly` [7]. It was built using `plotly`, so it was convenient to integrate the plot into our webpage. Since the Keyword popularity algorithm could be slow, we decided to form 4 pre-made plots, corresponding to a 1-month period, a 3-month period, a 6-month period, and a 12-month period. These are the period that we deduced users might be interested in.

3) Results: Figure 14 demonstrates how the plot on the final application would look like. In this figure, users can slide the slider to check the popularity in a specific period, or they can press the `Play` button for the plot to run by itself.

Some observations can be made regarding the plot. It seems that in most of the periods, the frequencies are typically less than 5, and when changing the period to 6 or 12, it is not uncommon to observe many words that have frequencies under 1. One reason we might think of is that the number of blogs that used the keywords might not have been written too much in that period. This is especially true for longer periods, as the total number of blogs is large, and the topics for those blogs are more diverse when compared to shorter periods.

D. Feature importance

Having extracted a wide range of features from the blog text (described in detail in section III), we sought to explore the extent to which each of these features affects the success of a blog. The following section directly addresses *RQ3*. To gain an insight into how well the features we extracted can serve as predictors for the success of the blog, we decided to train a series of regression models on all of the extracted

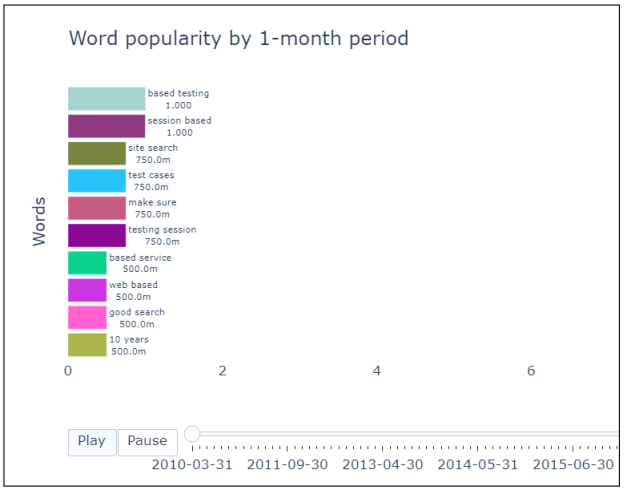

Figure 14: The keywords popularity plot

features. We then used the parameters of the resulting models as an estimation for the importance of the features.

1) Choice of models: Before we could proceed with training, we first needed to select the type of regression model to train, as well as the target variable that the model would predict. For the model types, we have elected to use a Linear Regression model (LR) [27] and a Random Forest Regression model (RFR) [28] from the scikit-learn library, as each of these models offers easily accessible importance weights for the predictor features. For the target variables, we have decided to use each of the four metrics obtained from Google Analytics: pageviews, average reading time, exit percent, and bounce rate. Since the extracted blog features could have different effects on each of the success metrics, we have trained a separate model for predicting each of the metrics. As a result, we trained 8 models – 4 LR models and 4 RFR models. We then extracted the importance weights from the models. Even though our goal was to obtain importance weights and did not include coming up with actual predictions for the target variables, we decided to measure the Root Mean Square Error (RMSE) of each model through cross-validation. The computed errors were then used to select the model that is better capable of predicting the target variable, and thus provides more reasonable importance weights for the features.

2) Results: The RMSE of the four models is reported below, along with the widths of the 90% quantile ranges:

Model	Target variable	RMSE	90% quant. range
LR	pageviews	1564.85	3896.25
LR	avg time on page	191.03	585.03
LR	bounce rate	0.23	0.65
LR	exit percent	0.16	0.56
RFR	pageviews	2068.26	3896.25
RFR	avg time on page	165.28	585.03
RFR	bounce rate	0.23	0.65
RFR	exit percent	0.15	0.56

TABLE I: Model Errors

Based on the observed errors, we can conclude that neither model was remotely successful at predicting the pageviews

value, as the error is large compared to the width of the 90% quantile range. This is an expected result, as users make a decision to click on a blog post before they see the contents of the blog, while the features we extracted are based on the qualities of the blog text. We then should not consider the importance weights reported by the models to be reliable when it comes to predicting pageviews. We can also observe that the RFR model outperforms the LR model in all categories except pageviews. It can then be assumed that importance weights extracted from the RFR models are more representative of the true importance of the features, compared to the importance weights extracted from the LR models. Finally, the resulting importance weights are reported in **[REDACTED]**.

The table below reports the top 3 most important features for each of the success metrics: **[REDACTED]**.

E. Online blog text analysis

In addition to the conclusions of the conducted analysis, we also wanted to provide users of the web page with an opportunity to gain insight into how their next blog might measure up to the existing blogs. To that extent, we have developed a simple tool for extracting the features that we used in our analysis from user-inputted text and displaying the results to the user in an at-a-glace form. In addition to providing numerical values of each of the features and an explanation of how the value was calculated, the tool also outputs where the values stand in comparison to existing successful blogs. In addition, the tool indicates if a feature is relevant to the success metric of the user's choice based on feature importance analysis. Examples of the tool output can be found in the appendix of this report.

F. Exploratory visualization

As we cannot anticipate every question or decision that might come up during the blog writing process (*RQ4*), we have decided to construct a general-purpose visualization tool to help writers explore the trends within existing data. We designed the tool to be as flexible as possible, with configurable axes, several data filtering options, as well as an option to examine select regions of the feature space in more detail. The tool was also augmented with a visualization of trend lines within the data, which were constructed using a Locally Estimated Scatterplot Smoothing (LOESS) [29]. Examples of the tool's design and output are included in the appendix of this report.

V. Ethical Issues

In recent years the possible ethical issues with big data, machine learning, and artificial intelligence have garnered more attention, and slowly the regulations and guidelines for data usage are following. Although our data were mainly from publicly available sources, we needed to take into account some potential ethical problems regarding the data, removal of past blogs, and the text analysis tool, which are covered in more detail in the following.

A. Data

All sources of data were freely available for non-commercial use. Any private data, such as the analytical metrics of Futurice's Blogs, was used after the signing of appropriate Non-Disclosure Agreements of all involved parties with the team. The code is hosted in a private repository, accessible only to the people that worked on the project. No personally identifiable information is used/stored by the project team.

B. Removal of Past Blogs

At the beginning of the project, the team did not consider what should happen in the event that a past blog post is deleted, for instance, in the case where the information presented in the blog is no longer accurate, or the author's personal views have changed. In this case, the downloaded blog would have to be removed from our dataset. Each blog has a non-trivial impact on our findings and conclusions, so our models would have to be re-run before the results can reflect the deletion of the blog.

C. Online blog text analysis

The online blog text analysis tool provides statistical comparisons between the inputted body of text and past successful blogs. There is a possibility that the blog writer could feel compelled or pressured to strictly optimize their blog based on the output of the tool. Although the tool was created with the intention of giving useful metrics to users, the team feels that there is a chance that it could impact the creative freedom of the author.

VI. Conclusions

For this project, we have approached the problem of leveraging the historic data about the blogs written by Futurice to help writers produce more engaging blogs from several directions. We extracted relevant textual features from the blog texts, explored the topic structure of the blog corpus, investigated alternative approaches to grouping and categorizing the blogs, visualized trends in keyword popularity, estimated the importance of extracted features for blog success, and constructed a simple recommendation system for blog writing. As the output of the project, we have produced the discussions detailed in this report, as well as a deployable visualization tool. The screenshots of the tool are included in the appendix of the report.

VII. Future Prospects

A. Scraping

Some further improvements we could make would include speeding up the code and having better exception handling. Currently, exceptions are just ignored, since they comprise a very small portion of the total number of blogs. More preprocessing could be done to clean the few very unique blogs (with tabular data or embedded elements.)

B. Topic Modeling

Currently evaluation of the quality of the topic model is rather crude and relies heavily on human judgment. Future improvements could include more quantitative comparisons between different topic models, for example with OCTIS [16] or some interpretation-based approaches [17].

C. Keywords popularity

The preprocessing for the keywords popularity algorithm did not filter out words that are in another language. So one future improvement for this part could be to find a suitable Language Detection model to integrate into the preprocess. We also want to improve the graph a bit. The graphing library that we used for the popularity graph does not have an animation when a word was taken over by another word. This can make the graph look confusing. So one future prospect is to find or create another animated graphing library. Since we pre-made the graphs and integrate the HTML file into the application, it would not be a big issue if the library can be a bit slow.

D. Additional feature extraction

While we were able to extract many valuable features from the blog data, there is a number of feature domains that remain unexplored. For instance, features based on blog titles or teasers could provide a better prediction of pageviews, as the users see the titles and teasers prior to deciding whether to click on an article or not. Another feature domain that could potentially provide valuable insights is the connectivity of the blog posts. Exploration of how blogs link to each other, and how such links affect the success of the blogs would be an exciting avenue for further work on this project.

VIII. Reflection

For many of us, this project was the biggest data science project so far, which means that we had a lot of learning from zero. In the beginning, the scope of the project was rather broad and open-ended, so finding the goal and keeping the correct direction took some effort. Eventually, we got a mutual clear vision, of where we were heading to.

In the real world, data and the problems at hand are much more complicated and less well-defined than assignments given in school books. Handling the data requires tolerance for exceptions as the time period of the blogs spans over ten years, which means that the formats of the blogs often differ from one another. We needed to combine data from two sources, which did not cover the same periods. As a result, data collection required more work. Even after cleaning the data, gaining insights into the structure and hidden information is difficult and requires numerous attempts for us to choose the most suitable methodology. The exploration itself is a valuable learning experience.

During the project, we learned some new data science tools, sometimes not in the most pleasant way. We got to experience configuration dependencies and their illogical feeling errors. Also, different python libraries and their all own seemingly similar but different implementations of DateTime-type made us scratch our heads at times. Still, maybe the most important skill for data scientist work is to be able to communicate results to an audience, which means the ability to create informative visualization and explain your findings.

IX. Contributions

While much of the work for this project was carried out through tight cooperation between multiple team members, each of us still ended up focusing on certain segments of the project more than others. The focus points are listed below:

- Duong Le
  - Feature extraction for the text length, average text length, and the readability score
  - The keyword popularity section in the application
  - Pre-processing for Google Analytics
- Georgy Ananov
  - General visualization section for the web application
  - UI for the online blog text analysis tool
  - Pre-processing for Google Analytics
  - Web tool deployment
  - Administrative tasks
- Guting Huang
  - Document exploration with topic modeling
  - Topic modeling section for the web application
- Laura Talvio
  - Feature extraction for sentimental analysis and lift score
  - Scraping URLs and a rough first iteration data from blog site
- Mael Chauvet
  - Research and Training of Clustering algorithms
- Rajat Kaul
  - Web scraping
  - Research into automation of scripts

References

[1] Grootendorst, M. (2022). BERTopic: Neural topic modeling with a class-based TF-IDF procedure. arXiv preprint arXiv:2203.05794.

[2] Bird, S., Klein, E. and Loper, E. (2009) "Accessing Text Corpora and Lexical Resources," in Natural language processing with python. Beijing: O'Reilly, pp. 39–78.

[3] Grootendorst, M.P. (2021) Frequently asked questions, FAQ - BERTopic. Available at: https://maartengr.github.io/BERTopic/faq.html (Accessed: December 8, 2022).

[4] Naushan, H. (2020) Topic modeling with Latent Dirichlet allocation, Medium. Towards Data Science. Available at: https://towardsdatascience.com/topic-modeling-with-latent-dirichlet-allocation-e7ff75290f8 (Accessed: December 8, 2022).

[5] McInnes, L. (2018) Basic umap parameters¶, Basic UMAP Parameters - umap 0.5 documentation. Available at: https://umap-learn.readthedocs.io/en/latest/parameters.html (Accessed: December 8, 2022).

[6] https://www.nltk.org/index.html

[7] Chaves, L. (2022) Raceplotly, PyPI. Available at: https://pypi.org/project/raceplotly/ (Accessed: December 8, 2022).

[8] Sklearn.cluster.kmeans (no date) scikit. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (Accessed: December 8, 2022).

[9] Sklearn.cluster.AgglomerativeClustering (no date) scikit. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html (Accessed: December 8, 2022).

[10] Sklearn.manifold.TSNE (no date) scikit. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html(Accessed: December 8, 2022).

[11] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[12] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[13] Chall, J.S. and Dale, E. (1995) Readability revisited: The new Dale-chall readability formula. Cambridge, MA: Brookline Books.

[14] McClure, G.M. (1987) "Readability formulas: Useful or useless?," IEEE Transactions on Professional Communication, PC-30(1), pp. 12–15. Available at: https://doi.org/10.1109/tpc.1987.6449109.

[15] https://maartengr.github.io/BERTopic/getting_started/guided/guided.html

[16] https://github.com/MIND-Lab/OCTIS

[17] http://users.umiacs.umd.edu/~jbg/docs/nips2009-rtl.pdf

[18] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[19] (no date) NLTK. Available at: https://www.nltk.org/howto/sentiment.html (Accessed: December 8, 2022).

[20] (no date) Google trends. Google. Available at: https://trends.google.com/trends (Accessed: December 8, 2022).

[21] GeneralMills (no date) Generalmills/Pytrends: Pseudo API for google trends, GitHub. Available at: https://github.com/GeneralMills/pytrends (Accessed: December 8, 2022).

[22] Grootendorst, M. (2020). Keyword extraction with bert, Medium. Towards Data Science. Available at: https://towardsdatascience.com/keyword-extraction-with-bert-724efca412ea (Accessed: December 8, 2022).

[23] Carbonell, J. and Goldstein, J. (no date). The use of MMR, diversity-based reranking for reordering documents and Procuding Summaries. Language Technologies Institute, Carnegie Mellon University. Available at: https://doi.org/10.1145/290941.291025 (Accessed: December 8, 2022).

[24] Reimers, N. and Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv preprint. Available at: https://arxiv.org/abs/1908.10084

[25] Basic umap parameters (no date) Basic UMAP Parameters - umap 0.5 documentation. Available at: https://umap-learn.readthedocs.io/en/latest/parameters.html (Accessed: December 8, 2022).

[26] Uniform manifold approximation and projection for dimension reduction (no date) UMAP. Available at: https://umap-learn.readthedocs.io/en/latest/index.html (Accessed: December 8, 2022).

[27] Documentation for Linear Regression. Scikit-learn. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (Accessed: December 8, 2022)

[28] Documentation for Random Forest Regressor. Scikit-learn. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. (Accessed: December 8, 2022)

[29] Documentation for the LOESS fit visualization. Altair. Available at https://altair-viz.github.io/user_guide/transform/loess.html#user-guide-loess-transform. (Accessed December 8, 2022).

APPENDIX

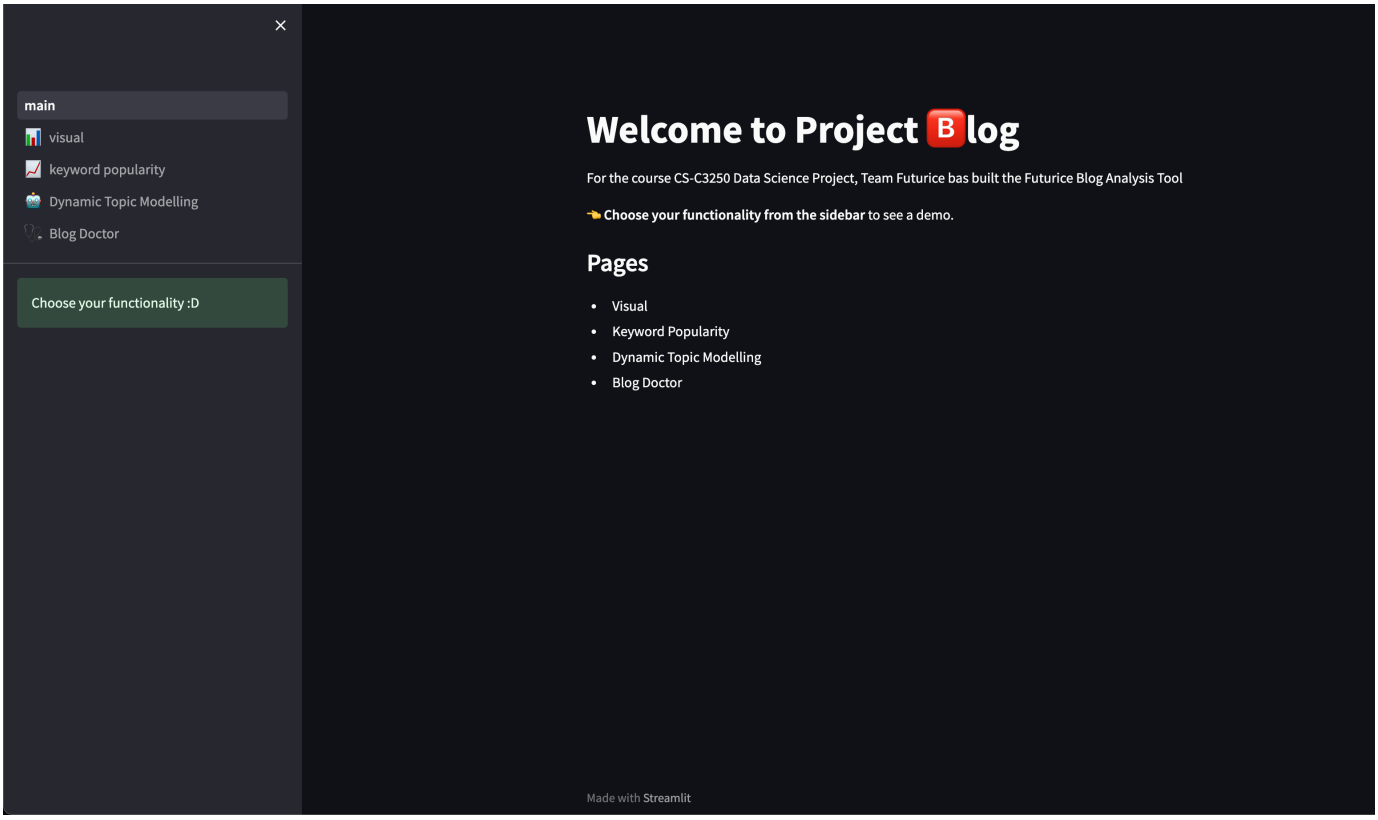

Figure 15: Home page with subpage navigator in left sidebar

Figure 16: Visual subpage, filters for the general data exploration tool

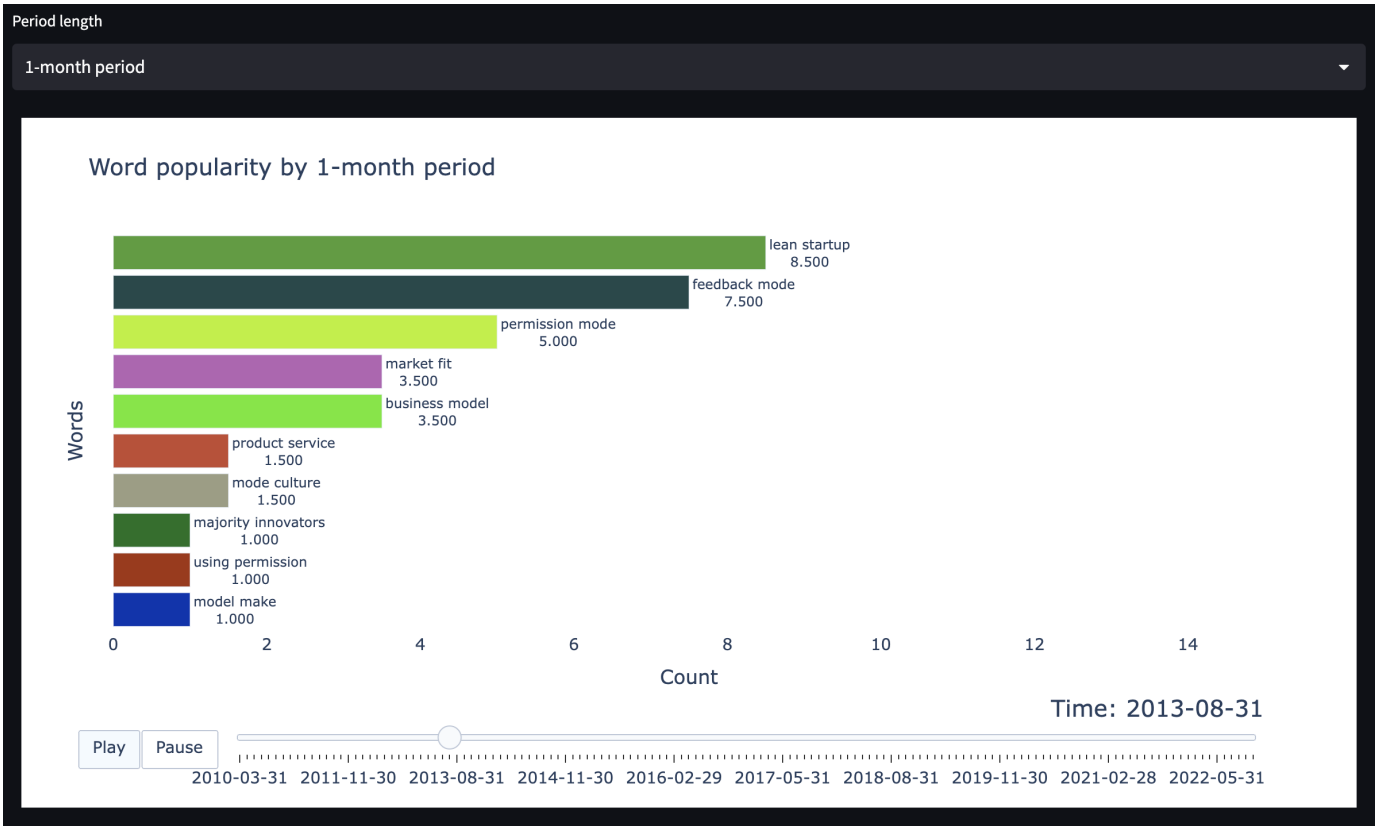

Figure 17: Keyword popularity subpage, animated keyword popularity visualization

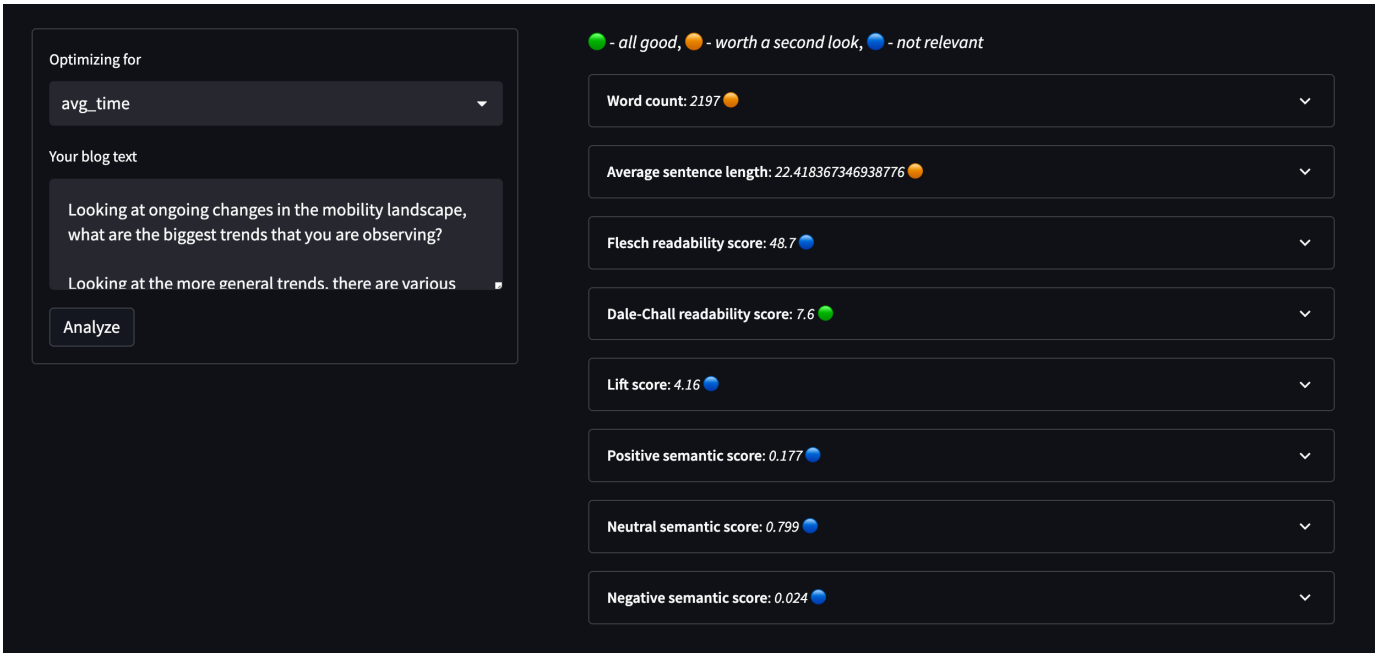

Figure 18: Blog Doctor subpage, online blog text analysis and recommendation system

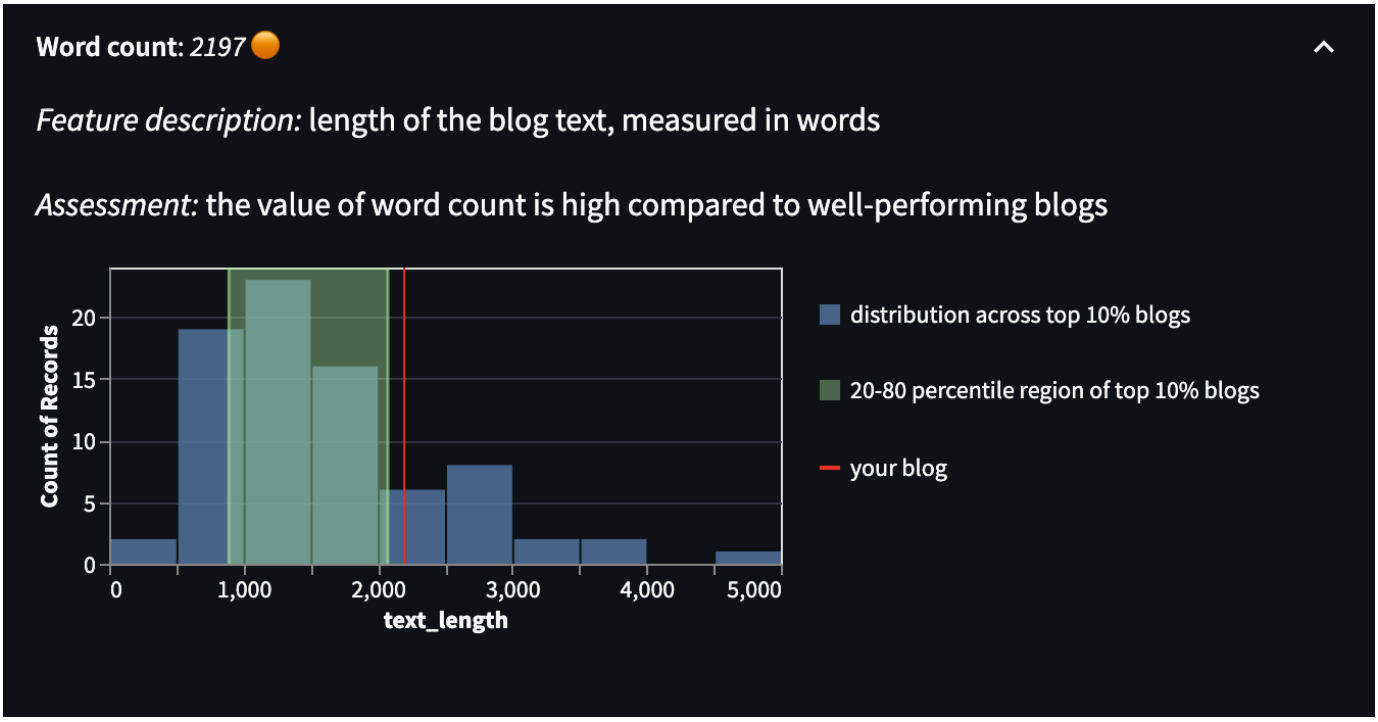

Figure 19: Blog Doctor subpage, online blog text analysis and recommendation system - expanded view example