CMPT732 Group Project Report

(http://www.webstylegenerator.com/)

By: Matthew Canute, Young-Min Kim, Donggu Lee, Adriena Wong

Problem definition:

What is the problem that you are trying to solve?

Colours and fonts have a huge impact on how visitors view online content, whether it's a company website or a personal blog. With so many possible colours and fonts, it's difficult to choose the perfect combination for a site. Our 'Web Style Generator' will recommend the perfect style by comparing a user's typical text content with other similar websites or blogs. Additionally, we wanted to look at possible colour or font trends within certain categories of websites and blogs using the same datasets.

What are the challenges of this problem?

After deciding on this problem space, there were several challenges that were considered before beginning the project implementation.

One of the challenges is that scraping content from a website sequentially is generally a slow task. We knew that we would have to make several requests to each website to scrape for the colours, fonts, and text information from their CSS files. In order to prevent having our IP addresses blocked, we considered setting a self-imposed timer which would have likely become the main bottleneck to the rest of the project, especially since we intended to scrape a very large number of websites. However, given a list of websites and access to a cluster with many nodes, we realized that we could divide and conquer the list of websites to scrape, each with their own self-imposed timer. Fortunately, when we actually started scraping websites, we did not encounter this problem and so the timer was not necessary.

Another challenge that we were worried about was how we would obtain comprehensive lists of Tumblr blogs and websites to scrape. After a quick search online, it was clear that although there was a Tumblr API that was publicly available, there was no existing list of Tumblr blogs. We also were initially unable to find a reliable list of websites that covered a large variety of topics.

Methodology:

To start, we separated the data collection stage in two parts: one for websites and one for Tumblr blogs. We scraped a list of over 100,000 websites from the Alexa Top Websites dataset (https://www.alexa.com/topsites/category). We assumed that this list would be fairly

representative as a sample snapshot of web design trends that would persist for at least the next year. For that reason, we decided to treat this as a one-time manual data pull.

Next, we built a web scraper using Python's BeautifulSoup4 package to parse through the website's main page (if possible), to check whether any 'about-us' pages exist and to grab the text data from it, or otherwise to simply grab the text from the main page. Then it searched for any CSS stylesheet pages linked on that website and parsed the text from those as well, collecting all HEX codes for the 'background-color' as well as all font-family collections. Finally, we created a frequency table for the colours and fonts before inserting the three most frequent colours and fonts into a Cassandra database.

We wanted to store the text along with the colour/font schemes in order to later calculate the semantic similarities of different websites using Word2Vec, and to determine if there were any established patterns between certain website categories and particular styles.

Since we found that web scraping took too long, we tried to parallelize it as much as possible. We considered using PySpark for this as it worked nicely with Cassandra, and we thought that the distributed map-reduce method of inserting from web-scraping would be the fastest approach. However, in reality, after some testing, we found that spawning multiple threads in shell scripts was actually slightly faster.

At the same time, we wanted to similarly scrape a set of structured websites in the style of blogs. We discovered that the Tumblr API was fairly intuitive to use, specifically using their PyTumblr Python package. However, they did not have a convenient list of their complete Tumblr blogs directory. As a result, we built a recursive structure to collect a large list of Tumblr websites which started with a number of diverse blogs, and traversed through those blog's re-blogs, and those blog's re-blogs, and so on.

We wanted a nice interface in order to interact with the semantic similarity of different websites, so we built a Flask application that would allow us to interact with a word-embedding word mover distance calculation. The user would input a piece of sample text, and would then be able to see the colours and fonts of the closest websites associated with that sample text. The application generates a sample view of what that website or blog would look like with given the colours and fonts. In order to show the relative frequencies of the closest websites' colours and font families, we made the size of the suggested colours and fonts change accordingly.

We built these data pipelines under the assumption, it could become a scalable set of ETL packages that could be run by a scheduler, such as a simple cron job once a month to potentially capture the changing stylistic trends in website design over time. This monthly batch file can then fit the new corpus of website text along with their colour/font schemes, and then cache the memory of the word-similarity function that waits for user input. As of now, our program is using a large sample of websites and blogs from November 2019.

Briefly explain which tool(s)/technique(s) were used for which task and why you chose to implement that way.

- Cassandra: We chose to use Cassandra as it is easily available on the cluster. Also, we were originally testing a distributed method of crawling websites and hypothesized that cassandra-spark integration would be good for parallel inserts.
- Flask: The cached word corpus similarity functions were native to Python, and so building a website using a Python framework, such as Flask, would be good. We also felt that it would be a good opportunity to learn an employable skill.
- Spark: We used it to connect to Cassandra. We also tried testing to see whether it made web-scraping faster through the divide-and-conquer approach.
- Pandas: It is easy to use for data analysis.
- Gensim: It contains useful pre-packaged natural language functions for us to use.
- BeautifulSoup4: It is an easy tool for web-scraping.
- NLTK: We used this for certain natural language tasks such as stop-word lists and stemming Other natural language tasks.
- Word2Vec: Word embeddings allowed us to calculate semantic similarity between different websites.

Problems:

What problems did you encounter while attacking the problem?

Some of the problems we faced during the project include:

1. Web-scraping is messy since the CSS stylesheets are often formatted differently for different websites.
2. It was also difficult to isolate the useful text from the javascript-related text.
3. Parsing some websites could take up to several seconds each.
4. Tumblr API had limits in the number of requests per hour and per day.
5. Amazon had limitations in the number of requests (although we still ended up with a list of over 100k websites).
6. The original pkl file, which references large cached variables storing each website's points within the word embedding space, was too big. As a result, it took about 3 minutes to run each query.

How did you solve them?

1. Considered all the possible permutations of the CSS representation.
2. Specified a custom stop-word list.
3. Scraped websites in parallel with several computers.
4. Ran the Tumblr API multiple times each day.
5. Ran the Amazon scraping script separately for each category of website.

6. Reduced the size of the pkl file to 10% of the original size by constraining it to use only words within the English language.

Results:

What are the outcomes of the project?

We built a frontend website (http://www.webstylegenerator.com/) that generates style recommendations based on the user's input text for websites and blogs separately. It gives an immediate visual of how the colours and fonts would look together. Also, as a direct result of scraping websites and blogs, we obtained a really fascinating dataset that could be used for further analyses. For example, by periodically scraping websites or blogs in a similar way, website style preferences and trends could be tracked over time. Also, association rules between popular colour and font pairings could be analysed. Finally, colours used by certain industries could be examined as well. Overall, this could be generally helpful to marketers looking to design a website.

What did you learn from the data analysis?

We plotted the RGB values of the main three colours found on websites in a 256 x 256 x 256 graph. It was interesting to see that there were clearly clusters of colours that are typically unused. In general, all colours across the greyscale were widely used by both websites and blogs. Bold primary colours were popular for websites whereas for blogs, there was a larger variety of colours but it was clear that pastel colours were more popular.

What did you learn from the implementation?

From the implementation, we realized that web scraping is difficult. We used BeautifulSoup4 to parse through the website's main page (or about-page if it exists). We then grabbed all the hex colours and font families from the CSS packages. This was difficult to do due to the large amount of information collected as well as the widely varying formatting conventions used by different websites. By successfully completing the web-scraping, we were able to gain valuable experience and learned a lot about how to effectively parse useful information from a large amount of text.

We also discovered that parallelization of tasks sometimes can be faster by just spawning different threads rather than using Spark. After running some quick tests on web-scraping with and without Spark, we realized that using multiple threads was actually slightly faster compared to Spark.

Finally, it was our first time using Flask so there was a small learning curve at the beginning of the project when trying to set up the web frontend.

Project Summary:

Project Priorities Ranking:

Project Tasks	Score
Getting the data: Acquiring/gathering/downloading.	4
ETL: Extract-Transform-Load work and cleaning the data set.	3
Problem: Work on defining problem itself and motivation for the analysis.	1
Algorithmic work: Work on the algorithms needed to work with the data, including integrating data mining and machine learning techniques.	2
Bigness/parallelization: Efficiency of the analysis on a cluster, and scalability to larger data sets.	2
UI: User interface to the results, possibly including web or data exploration frontends.	4
Visualization: Visualization of analysis results.	2
Technologies: New technologies learned as part of doing the project.	2
Total	**20**