Software Design
Homework 5: Mining Text from the Web
Pratool Gadtaula and Greg Coleman

Project Overview

Our group attempted to build a program that maps a course from one specified Wikipedia page to another only through other hyperlinked Wikipedia pages. We approached this problem by imagining the path as a map. Each Wikipedia page links to many other Wikipedia pages. We wanted to find the next page by looking at all of the linked pages and comparing it to the target page. The one that was most similar would then be the next page.

Implementation

First, the our code passes in a starting and ending point, the Wikipedia pages that one must start and end on. All of the linked articles from the starting page are then analyzed based on the frequency of each word. All the words that contain fewer than 5 characters are ignored. The words and their corresponding frequencies are stored into a dictionary in which the keys are the words and the values are their frequencies. After all of the linked articles are analyzed in this way, reduced to a dictionary of words and their frequencies, the linked articles are compared to the frequency breakdown of the target article. This is done by making an even larger dictionary that contains all words longer than 4 characters between the two articles. Then, the values of each of the words is populated with their corresponding frequencies. This larger dictionary contains keys that one of the smaller dictionaries do not have. In these cases, the corresponding value is 0. The frequencies of each of the articles are then compared by taking the cosine similarity measure of these two frequency vectors. This will relate to a value between 0 and 1, where 1 indicates that the two articles' word frequencies are exactly the same. The hyperlinks of each of the linked Wikipedia pages from the starting article are also compared by counting each of the hyperlinked Wikipedia page that the starting article contains that are also contained in the final article. This value is then divided by the total number of hyperlinks in the ending article, resulting in a value between 0 and 1. These two analyses are combined by average the correlation values. The wikipedia page that is linked to the starting article that has the highest correlation is the next article that is examined with respect to the target article. This process is repeated until the article being examined is the target article. At that point, a list with the article titles that the program went through in order from start to target article, is returned.

Results

When trying to find a path between Basil McRae and the New York Rangers, the program output the following:
['Basil McRae', 'Toronto Maple Leafs', 'New York Rangers']
Coincidentally, this is also the shortest path. There are also flaws to this algorithm. One example is that if it detects two equally relevant articles based on their hyperlink similarity, it takes takes the second article, even if the first article is the target article. This will cause a roundabout issue just like the one below:
['Basil McRae', 'Toronto Maple Leafs', 'Ice hockey']
In this case, Basil McRae and Ice hockey are directly linked. However, Toronto Maple Leafs ends up having the same correlation and comes second. So instead of getting a direct link, a path with one extra page is taken.

Reflection

Overall we were happy with how our project went. Even though this is one of the first times we coded in a group, in general we were able to work efficiently. Early on we made a outline of all the functions we needed and we splitted up the task from there, which worked really well. However, our work schedule

was drastically ill-planned and could have used much improvement. We greatly underestimated how long this project would take and we wished we started a little earlier. In the end it was not a huge deal because we did produce working code, but we had to simplify our original algorithm a little due to time constraints so our approach is not the most efficient. We feel like the project was appropriately scoped. Even though it took us a lot longer than we thought, our project idea was still realistic and we learned a lot about text analysis. We made sure we did unit tests for all of our main functions, in order to ensure they were robust. However, we had a hard time unit testing some of the functions because it was hard to calculate the expected value. For example, our compare_wiki function that rearranges two dictionaries in the same order as each other, in order to find the cosine similarity, we could not predict how python would rearrange those dictionaries. For cases like these we just checked to see if both dictionaries were in the same order as each other and did not try to predict the outcome.