DocSim

Mark Klein

March 11, 2014

1 Problem Overview

People from all over the world write research papers, novels, and code; however, all of this writing suffers the possibility of plagiarism. In fact, plagiarism in school is more than prevalent at this time. In order to prevent this widespread issue, plagiarism checkers are desperately needed. How should one work though? Do they check every word and compare word counts? Every other word? A sequence of words? Well, it depends on the piece of work in question. Different types of papers will be plagiarized in different ways. Sometimes just a sentence will be taken word for word. Sometimes the entire paper will be. Because of the large number of different ways someone can plagiarize, the ill effects it has on students' ability to learn, and the loss of original ideas plagiarism creates, it is important to develop an application that can effectively identify plagiarism.

2 Development Process

2.1 Ideating

The development process consisted of three main phases: ideating, coding, and testing. The ideation phase was first and in this phase I determined the best way to identify plagiarism. The optimal program would not only identify plagiarism with a high degree of certainty, but also be able to compare documents quickly using a relatively low amount of memory. The constraints on time and memory that is available for the program to use were the hardest to get around because the most effective way to identify plagiarism would be to compare every phrase in one document to every phase of another document. This method proved to take much too long and use way too much memory, so this idea was quickly scrapped.

Another idea that came to mind was to do a phrase count of each document. In other words, the phrases that appeared in each document would be tallied and compared, which would take linear time instead of quadratic time; this would be a huge improvement. The problem with this is method is that there is no clear way of identifying plagiarism. Of course documents that have more phrases in common show signs of plagiarism, but there's no certain way of identifying this plagiarism because longer documents would

generally share more phrases just by random chance. This flaw demonstrated that simply doing a phrase count would not be the right approach.

The method chosen was to use cosine similarity to find the similarity of two documents. Essentially, this means translating the documents into vectors and finding the angle between the vectors. If the angle is 0°, then the documents are identical, and if the angle is 180° then the documents are completely different. This algorithm can be executed in linear time, making it just as fast as the previous method discussed, but it is much more accurate. The math behind this is discussed more in section 3.

2.2 Coding

The coding section of creating this application was relatively brief. Because the algorithms that needed to be implemented were fairly trivial, figuring out how to write them wasn't that tough and didn't take much time; however, because java is still a fairly new language to me the coding wasn't as quick as it could have been. Unlike other languages that have a simple *read*() function, java requires a *Reader* object to be created, which is used to read a file line by line. As can be seen by my commits, figuring out how to read in the file took the longest because the math was simple to code.

2.3 Testing

After the initial coding was completed, it was time to test the almost-finished product. At first it seemed as though everything was going smoothly; however, I soon realized that I wasn't comparing every n-th word, but every n-th character. Luckily, it was a simple problem with a regex string I was using, and not a tremendous flaw in the code I had written. Nextly, I found that it would be beneficial to explain how the comparison was being done in the product so I added an optional explain flag that could be passed in through the command line when the program was being run. I also found that allowing a specific directory to be chosen and adding an optional help flag would be useful as well.

3 Solution

3.1 Overall Result

Overall, the result is very useful. It successsfully compares documents and clearly displays the cosine similarity between them. The program has a useful help feature and is user friendly. This all contributes to it being helpful and a product that can be used by anyone to detect plagiarism and promote a plagiarism-free environment.

3.2 The Math Behind Everything

To see how cosine similarity works, let's examine two strings:

A) "The cat ran"
B) "The dog ran"

Strings A and B are then split into case-insensitive arrays of three strings each.

A) {"the", "cat", "ran"}
B) {"the", "dog", "ran"}

Then, the arrays are converted into hashtables with each word as the key, and the number of times each word appears as the value.

A) {"the": 1, "cat": 1, "ran": 1}
B) {"the": 1, "dog": 1, "ran": 1}

The hashtables are then merged and sorted into alphabetical order by key.

A) {"cat": 1, "dog": 0, "ran": 1, "the": 1}
B) {"cat": 0, "dog": 1, "ran": 1, "the": 1}

Now, the values are ignored and only the numbers are considered.

A) {1, 0, 1, 1}
B) {1, 1, 0, 1}

These two arrays are then acknowledged as vectors, and the angle between these vectors is used to determine the similarity of the documents.

Let θ = the angle between the vectors,
And $\mathbf{u}$ and $\mathbf{v}$ be the vectors A and B respectively. So we have

$$\cos\theta = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \times ||\mathbf{v}||}$$

Going back to the example, we plug in the numbers to get

$$\frac{1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1}{\sqrt{1^2 + 0^2 + 1^2 + 1^2} \times \sqrt{1^2 + 1^2 + 0^2 + 1^2}} = \frac{2}{3}$$

This means that the cosine similarity between the two strings is $\frac{2}{3}$. The more similar two documents are, the closer to 1 the cosine similarity of those documents are. In other words, if two documents are the same, their cosine similarity will be 1 and if they are compeltely different, their cosine similarity will be 0. The program uses this algorithm and outputs the similarities between every document in a specified formula. The program can be run using the jar file included in this repository.