Practical 5

Question 1:

*Katherine Heigl is an American actress who is 38 years old and has won an Emmy*

*Catherine ZetaJones is a Welsh actress who is 48 years old and has won multiple Oscars*

*Katharine Hepburn was an American actress who died at age 97 and had won multiple Oscars*

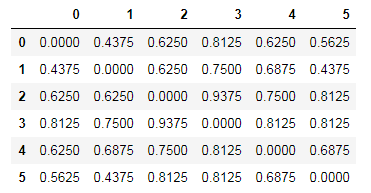
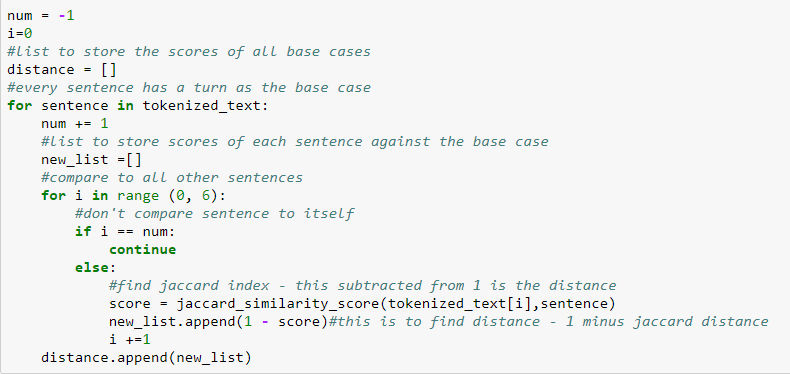
*Catherine Bell is a 48 year old American British Iranian actress who’s never won an award*

*Catherine Keener is an American actress who is 58 and has won a group SAG award*

*Catherine McCormack is a British actress who is 45 years old and is famous for theatre*

1. The entities I chose to describe were famous Katherines. I picked six and wrote a brief bio about them:

Figure 1: The sample text used in question one (originally a text file)

The instructions were to modify the jaccard\_index python program to become jaccard\_distance, but unfortunately, this particular code worked with strings whereas my text items are being read in as lists. Because set is not a property nor function of lists, I had to write my own jaccard distance code by using the jaccard\_similarity\_score from sklearn.metrics. The similarity score returns the jaccard coefficient which represents dissimilarity. So when subtracted from 1, it yields the jaccard distance, which was calculated in the code (seen figure 2). The subsequent distances are shown in fig 3. The next task was to show empirically that the property of Fig 2: code to find jaccard index and distance triangle inequality holds for this measure. The property of triangle inequality can be explained as follows: In geometry, the sum of the length of two sides of any triangle must be greater than the length of the third. So in triangle ABC, the sum of length A and length B must be greater than length C. This property can apply to distances as well, when written with vectors (Triangle inequality, 2017). Any distance metrics space must have this property – specifically, the sum of any two pairwise distances should be greater than a third pairwise distance. In this case, the sum of any two Jaccard distances from a Fig 3: Table of Jaccard Distances for the text items single base case must be greater than any of the distances alone. Code to discover and prove this can be seen in Figure 4. As can be seen, the property holds.

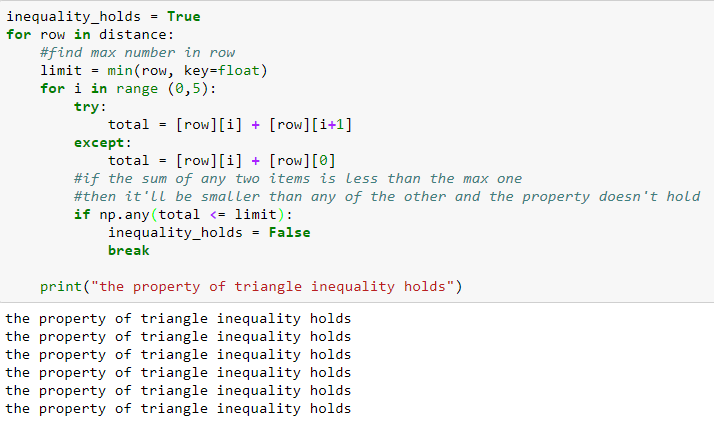


Fig. 4: code to test if triangle inequality holds with jaccard distance

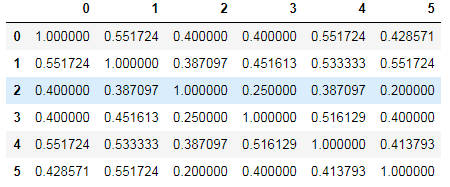
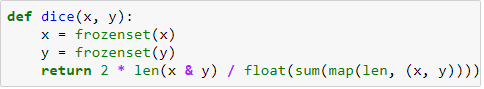
1. The Sorensen/Dice coefficient was found by establishing a dice function inspired by a function written by siguniang on their wordpress (Siguniang, B., 2015) (figure 5) and then applying it to the code found in figure 2 replacing jaccand\_similarity\_index with dice. It also measures the distance as 1 minus the dice score. The function follows the math necessary to find dice coefficient exactly. The table of the scores can be found in figure 6.

Fig. 5: function to find dice coefficient

It was then tested to see if triangle inequality holds with dice coefficients. Dice coefficients Fig. 6: Table of Dice distances for all text items aren’t a true measure of distance, so the property doesn’t have to hold. With the same data set used, it still held, as the differences between the texts still allowed this property to be retained. But with changing item number 5 from “Catherine Keener is an American actress who is 58 and has won a group SAG award” to “Catherine Keener is a 58 year old actress who has recently won a group SAG award” this gives it a similarity of 0 with item number two. This similarity of zero, and hence a distance of 1, will be very difficult for some sums of two dice distances to be greater than. Hence, the triangle inequality property will be broken. However, despite this logic and effort, the triangle property still held. To try to break it further, I changed the sentence to something completely different: “Sushi Keener bacon twelve 2 4 5 dog cat has recently now oops group SAG tabletop”. Unfortunately even with this completely different text set, the property still holds. This is probably because the dice coefficient is forgiving and accepting of heterogeneous items and fuzzy data (data linked by degrees of similarity). The way it is calculated, possibly due to the way it employs bigrams, allows it to still find some similarity between the two texts despite the completely different words.

Question 2:

*Advanced Grammar for new English as a Foreign Language Teachers*

*Learning Teaching: the Essential Guide to Foreign Language Teaching*

*English Language Teaching Today: Linking Academic Theory and Practice*

1. The documents tested were textbook titles I found at the school where I work.

The variants of document 3, all of which were designed to test something specific, are as follows:

*a. Spanish language teaching today: linking academic theory and practice*

*b. Advanced Spanish language teaching linking academic theory and practice for new teachers*

*c. Spanish language learning the essential guide to academic theory and practice*

*d. English language today: theory and practice for foreign language teachers*

*e. Language Today Teaching: Linking Practice Academic and Theory English*

**Variation A** tests the influence the word ‘English’ has on the documents. By changing ‘English’, a word found twice in the corpora to ‘Spanish’, a word found once and lowering the instances of ‘English’ from two to one will change the tf/idf scores of the words and thus change their effects on the cosine similarity. Because all other parts of the text are equal, this variation will isolate and test this phenomenon. It also does not have the word ‘teacher’ like document 1, only ‘teaching’ like document 2.

**Variation** **B** adds the word “Advanced” to share more tokens with document 1 and also replaces “Spanish” with “English”. The first was done to see if a shared word will heighten the cosine similarity or bring it down because it effects the tf/idf. The second was to mimic variation A but also to see if anything would change because now it shares the words ‘teachers’ and ‘advanced’ with variation A. Perhaps

**Variation C** puts two words in common with document 2, to test how the addition of multiple (more rare) shared words effect the cosine. It also removed any variations of the word “teach” replacing with “learn” to see if the semantics are marked differently or if those are considered “close” words (as in they are opposites) and if that changes the cosine.

**Variation D** has no instance of the word “teaching” only “teacher” this

1. The cosines between d1 and d2 and all other documents can be seen in figure seven found with using the code given in class. Most noteworthy, I thought, was the results between d2 and d3e – although they had two words that were similar, they still had a cosine of zero – as far apart as they possibly could be. I believe this illustrates the idea of cosine similarity and the integration of tf/idf perfectly: the reason that the two identical words “language” and ‘learning” did not cause a higher cosine of similarity is that these words appear often in this tiny corpus being examined and thus, their presence doesn’t carry as much significance or weight and hence why they didn’t have much of an impact on the cosine. Finally, Figure 8 shows a table of the cosine equivalent in degrees as calculated with the math library and the code cosine = get\_cosine(text1, text2) , angle\_in\_radians = math.acos(cosine) (math.degrees(angle\_in\_radians)). This was found because the hope was to somehow find a way in python to create one of those vector graphs where the cosine is represented as the angle between the vectors. I endeavored to find that angle, and, Fig 7. A Chart of the cosine differences between the various documents given more time, I look forward to finding a way to apply it to an information visualization.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d1 | 0 | 88.854 | 88.338 | 90 | 83.915 | 90 | 87.019 | 87.306 |
| d2 | 88.854 | 0 | 88.624 | 88.73 | 88.68 | 79.16 | 8.51 | 90 |

Fig. 8: The Cosine similarities between the documents translated into degrees

1. The library used was sklearn.metrics.pairwise and the code used was helped via the Dariah-De online tutorial (Dariah-De, n.d.).

Works Cited

Dariah-De. (n.d.). Working with text. Retrieved October 16, 2017, from https://de.dariah.eu/tatom/working\_with\_text.html

Siguniang, B. (2015, December 30). (Jaccard/Simpson/Dice)Python. Retrieved October 14, 2017, from https://siguniang.wordpress.com/2015/12/30/similarity-index-of-jaccard-dice-overlap/

Triangle inequality. (2017, October 19). Retrieved October 15, 2017, from https://en.wikipedia.org/wiki/Triangle\_inequality