Practical 5

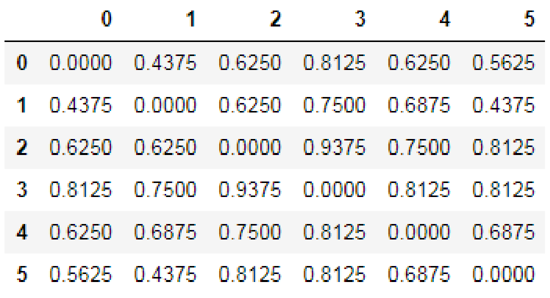
Question 1:

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

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

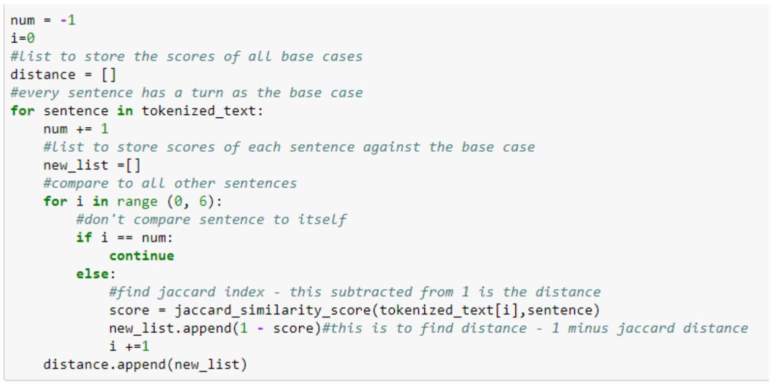
Figure 1: The sample text used in question one (originally a text file)

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 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 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.



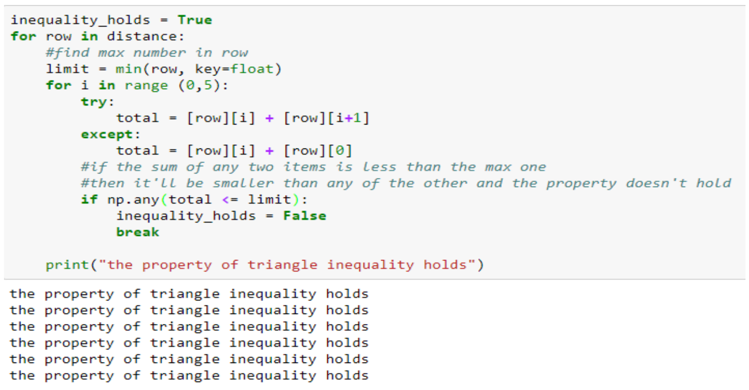
	0	1	2	3	4	5
0	0.0000	0.4375	0.6250	0.8125	0.6250	0.5625
1	0.4375	0.0000	0.6250	0.7500	0.6875	0.4375
2	0.6250	0.6250	0.0000	0.9375	0.7500	0.8125
3	0.8125	0.7500	0.9375	0.0000	0.8125	0.8125
4	0.6250	0.6875	0.7500	0.8125	0.0000	0.6875
5	0.5625	0.4375	0.8125	0.8125	0.6875	0.0000

Fig 2: code to find jaccard index and distance



```python
num = -1
i=0
#list to store the scores of all base cases
distance = []
#every sentence has a turn as the base case
for sentence in tokenized_text:
    num += 1
    #list to store scores of each sentence against the base case
    new_list =[]
    #compare to all other sentences
    for i in range (0, 6):
        #don't compare sentence to itself
        if i == num:
            continue
        else:
            #find jaccard index - this subtracted from 1 is the distance
            score = jaccard_similarity_score(tokenized_text[i],sentence)
            new_list.append(1 - score)#this is to find distance - 1 minus jaccard distance
            i +=1
    distance.append(new_list)
```

Fig 3: Table of Jaccard Distances for the text items



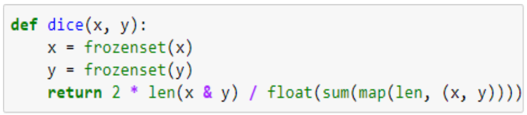
```
inequality_holds = True
for row in distance:
    #find max number in row
    limit = min(row, key=float)
    for i in range (0,5):
        try:
            total = [row][i] + [row][i+1]
        except:
            total = [row][i] + [row][0]
        #if the sum of any two items is less than the max one
        #then it'll be smaller than any of the other and the property doesn't hold
        if np.any(total <= limit):
            inequality_holds = False
            break

    print("the property of triangle inequality holds")
```

```
the property of triangle inequality holds
the property of triangle inequality holds
the property of triangle inequality holds
the property of triangle inequality holds
the property of triangle inequality holds
the property of triangle inequality holds
```

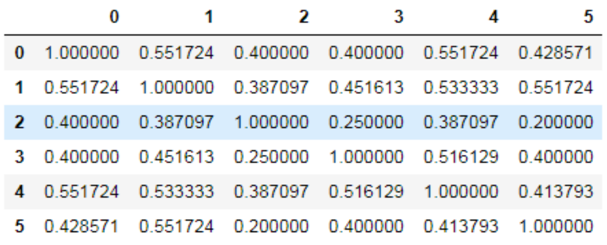
Fig. 4: code to test if triangle inequality holds with jaccard distance

b.) 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.



```
def dice(x, y):
    x = frozenset(x)
    y = frozenset(y)
    return 2 * len(x & y) / float(sum(map(len, (x, y))))
```

Fig. 5: function to find dice coefficient



	0	1	2	3	4	5
0	1.000000	0.551724	0.400000	0.400000	0.551724	0.428571
1	0.551724	1.000000	0.387097	0.451613	0.533333	0.551724
2	0.400000	0.387097	1.000000	0.250000	0.387097	0.200000
3	0.400000	0.451613	0.250000	1.000000	0.516129	0.400000
4	0.551724	0.533333	0.387097	0.516129	1.000000	0.413793
5	0.428571	0.551724	0.200000	0.400000	0.413793	1.000000

Fig. 6: Table of Dice distances for all text items

It was then tested to see if triangle inequality holds with dice coefficients. Dice coefficients 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:

a.) The docs tested were textbook titles I found at the school where I work.

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

The variants of doc 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 learning 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 docs. 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 doc 1, only 'teaching' like doc 2.
Variation B adds the word "Advanced" to share more tokens with doc 1 and replaces "Spanish" with "English". The first was done to see if a shared word will increase 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 doc 2, to test how the addition of multiple (more rare) shared words effect the cosine. It also replaced any variations of the word "teach 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 the word "teacher" (like doc 1) and the word "learning" (like doc 2). The other words are either found in both docs or neither. This variation will indicate if these words have different weighted influence on the cosine. I suspect that there will not be a difference as both examples of words will appear in the corpora the same amount of times and in an individual text the same amount of times, thus yielding the same TF/IDF scores. The TF/IDF score is a key component of calculating cosine similarity. Therefore, I hypothesize that the similarity cosines for doc 1 and 2 with this variation will remain the same in proportion to one another as those scores with the original doc three. In other words, the scores may change, but the difference between the two will not.
Variation E has the same words but a different order as the original doc three. I predict that this will not change the cosine as cosine depends on tf/idf and word order does not affect this metric. However, it is worth testing, as sometimes word order does matter (such as with distance metrics like dice or jaccard).

b.) The cosines between d1 and d2 and all other docs can be seen in figure seven found by using the code given in class and merely replacing the text associated with doc3 in def load_docs() and re-running the code. The experiment yielded some interesting results. In **variation A**, replacing the word 'English' with 'Spanish' had a significant effect on the cosine between this variation and doc 1, eliminating it almost entirely. This could be because d1 and d3 only share two words: 'English' and 'language'. All texts have the word

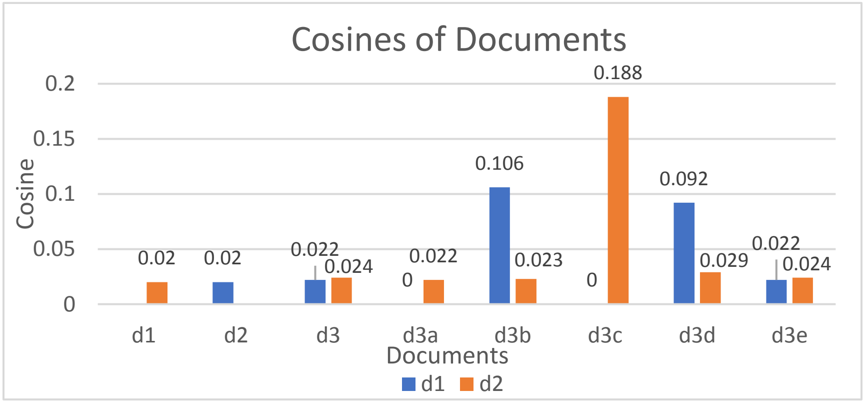


Fig 7. A Chart of the cosine differences between the various docs

'language' so that will probably have a lower weight than other words. English clearly carried a lot of weight because it's replacement was the difference of .022 similarity. This word change didn't affect doc 2 nearly as much, although the same change took place.

Variation B results indicate that shared words that are not found in all docs do positively affect cosine similarity. This is due these words' high tf/idf score.

Variation C indicates that "learn" and "teach" and their various forms are NOT grouped semantically, as doc 1 contains the word "teacher" but has a cosine of 0. This lack of grouping can probably be attributed to the minuscule data set not giving many examples of context and so words that don't appear together in the given docs will not be deemed similar. I also think it is interesting that although doc 1 and this variation shared the word "language", their cosine was still zero. The words are 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 word "language" appears often in this tiny corpus thus, its presence doesn't carry as much significance or weight, hence why it didn't have much of an impact on the similarity. So even if a word is shared, if it crops up often throughout the text, then it won't have much weight.

Variation D was interesting in that it shows that the word "teacher" in this corpus is weighted much more than the word "learning" as the sharing of teacher gave doc 1 a much higher similarity score than doc 2's sharing of "learning".

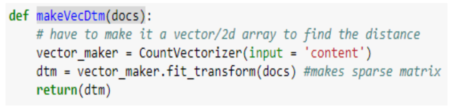
Variation E yielded the exact same results as the original doc three, thus proving that word order does not matter when it comes to calculating cosine similarity.

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, in python, create a vector graph where the cosine is represented as the angle between the vectors. I endeavored to find that angle, and, given more time, I look forward to finding a way to apply it to an information visualization.

c.) The library used was sklearn metrics pairwise Euclidean distances and cosine similarity. Functions written were inspired by the Dariah-De online tutorial (Dariah-De, n.d.). The first step was to make the docs into word vectors (code in figure 9) so that the Euclidean distances (figure 10) and cosines (figure 11) could be calculated. Docs 1 and 2 were compared to all variations of doc3 in both distance and similarity. The results (first example in figure 12) indicate that distance and cosine similarity are inversely proportional. So, when the cosine of similarity is 1 (identical) the distance is 0 and, vice versa, when distance is 4 (the maximum in this case, is 0 or completely different. This is because distance and similarity are opposites. Ergo, the higher one value, the proportionally lower the other, as illustrated by figure 12, where the first matrix starts with a 0, the other with a 1; the middle row ends with 3.7 in the first matrix and with a 0.3 in the second (.3 being the difference between 3.7 and the next whole number).

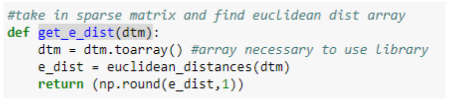
	d1	d2	d3	d3a	d3b	d3c	d3d	d3e
d1	0	88.854	88.739	90	83.915	90	84.721	88.739
d2	88.854	0	88.624	88.73	88.68	79.16	88.338	88.624

Fig. 8: table of the angles (in degrees) between the doc word vectors



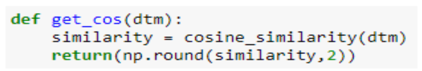
```
def makeVecDtm(docs):
    # have to make it a vector/2d array to find the distance
    vector_maker = CountVectorizer(input = 'content')
    dtm = vector_maker.fit_transform(docs) #makes sparse matrix
    return(dtm)
```

Fig. 9: Code to transform doc to vector and dtm



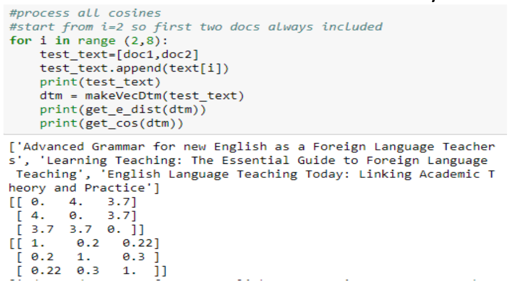
```
#take in sparse matrix and find euclidean dist array
def get_e_dist(dtm):
    dtm = dtm.toarray() #array necessary to use library
    e_dist = euclidean_distances(dtm)
    return (np.round(e_dist,1))
```

Fig. 10: Code to get Euclidean dist of two docs



```
def get_cos(dtm):
    similarity = cosine_similarity(dtm)
    return(np.round(similarity,2))
```

Fig.11 Code to find cosine similarity



```
#process all cosines
#start from i=2 so first two docs always included
for i in range (2,8):
    test_text=[doc1,doc2]
    test_text.append(text[i])
    print(test_text)
    dtm = makeVecDtm(test_text)
    print(get_e_dist(dtm))
    print(get_cos(dtm))
```

```
['Advanced Grammar for new English as a Foreign Language Teacher
s', 'Learning Teaching: The Essential Guide to Foreign Language
 Teaching', 'English Language Teaching Today: Linking Academic T
heory and Practice']
[[ 0.   4.   3.7]
 [ 4.   0.   3.7]
 [ 3.7  3.7  0. ]]
[[ 1.    0.2   0.22]
 [ 0.2   1.    0.3 ]
 [ 0.22  0.3   1.  ]]
```

Fig.12: Dist and Cosine matrix for doc1, doc2, and doc3

Question 3:

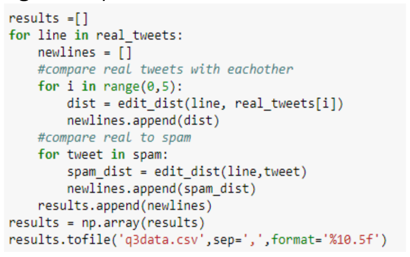
These are five normal tweets. Number five is from the infamous tweeter himself, Donald Trump:

1.Obama is the ex that keeps getting invited to events because everyone still loves him anyway #Ilovebarry
2. Yes I am a baby boomer and I still love that #pumpkinspicelatte! Guess I'm a #millennial at heart hah!
3. My uber says Jesus is arriving now in a Honda Accord. Not really how I imagined the second coming
4. The most disappointing thing about the UK election is there wasn't even a hint of Russian interference.
5. I have been saying for weeks for president obama to stop the flights from West Africa. So simple, but he refused. A TOTAL incompetent!

I picked number 5 to create spam tweets because trump's tweets are ridiculous. The first five are listed below:

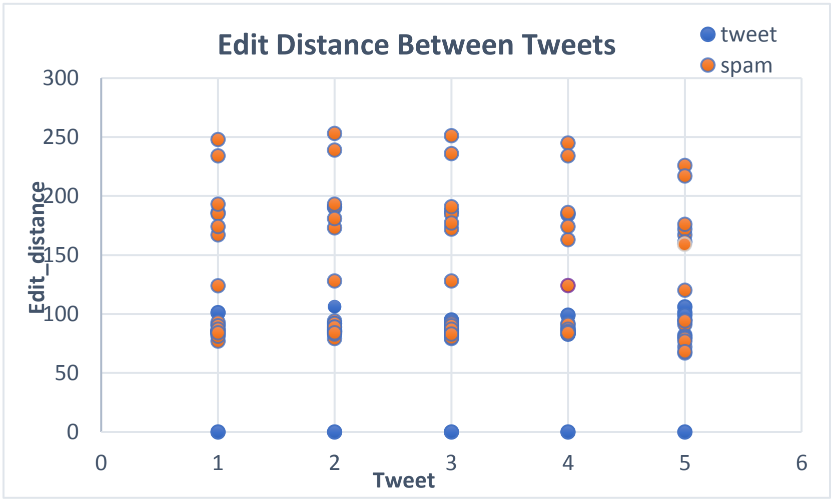
@Keypeele Tell president obama to stop the flights from West Africa! #NotoEbola #stoptheflights #travelban
@jjcampbell Tell president obama to Stop the flights from West Africa! Sign the petition! #notoebola https://www.change.org/p/barack-obama-ban-flights-from-west-africa-to-the-us-and-restrict-travel-to-the-us-from-ebola-stricken-areas
@Ihatejam why won't obama protect us? Stop the flights from West Africa! #ebola #notoebola #stoptheflights #travelban
@kcarrie obama cares more about saving face than protecting Americans! Stop the flights from West Africa! #notoebola
@PPrez Stop the flights! #ebola #notoebola #stoptheflights Sign the petition! https://www.change.org/p/barack-obama-ban-flights-from-west-africa-to-the-us-and-restrict-travel-to-the-us-from-ebola-stricken-areas

I used the edit distance code from lectures, but put it into a function. I then opened both the spam and real tweet files and calculated the edit distance between each real tweet and the other real tweets and spam tweets (code figure 13). This was then saved as a csv and plotted (figure 14). I understand that the idea is that the spam tweets will be grouped closely to one another because they will be quite similar. As can be seen from the provided sample spam tweets, this was the case. The only tweets above 150 are spam. The tweets with the highest edit distance (above 200) were those containing hyperlinks, which makes sense as the function would calculate the words within the link which would strongly differ from a tweet without a link. However, the spam falls in three clusters – those with a score of above 200, 150-200, and around the 80 to 110 mark. These groups are consistent, only varying in score slightly across the five tested real tweets. It is interesting that, where the average edit distance does vary is when it drops at tweet 5, the tweet inspiring the spam. Overall, most spam tweets were obvious. However, spam scoring around 80-110 fully overlap the scores of the real tweets, sometimes with even lower scores. This is quite interesting and I would love to learn more as to why this is. Finally, as a side note, it was also interesting that tweet two, the only one that contained two hashtags, had the second lowest edit distance. Considering that almost all spam messages also contained hashtags, I wonder if this has something to do



```
results =[]
for line in real_tweets:
    newlines = []
    #compare real tweets with eachother
    for i in range(0,5):
        dist = edit_dist(line, real_tweets[i])
        newlines.append(dist)
    #compare real to spam
    for tweet in spam:
        spam_dist = edit_dist(line,tweet)
        newlines.append(spam_dist)
    results.append(newlines)
results = np.array(results)
results.tofile('q3data.csv',sep=',',format='%10.5f')
```

Fig.13: code to find edit distance



with it. However, if this is the case, then I wonder why tweet one, with one hashtag did not also have a lower edit distance.

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