**Q1. 1) Make up your own set of word features describing 6 different entities; with some obvious overlaps and differences.**

**a. Modify the Jaccard-Index python program to do Jaccard-Distance and then compute all pairwise distances between the entities.**

**Based on results, show empirically, that the property of triangle inequality holds for measure.**

**Ans.** I have created a python dictionary ***target\_dic*** which consists 6 different targets(entities).

* **Pre-processed the dictionary tokenize -> Removal of stop words -> Removal of special words->Normalization -> Removing blank spaces -> Lemmatization -> Joining elements back -> Final Processed Dictionary.**

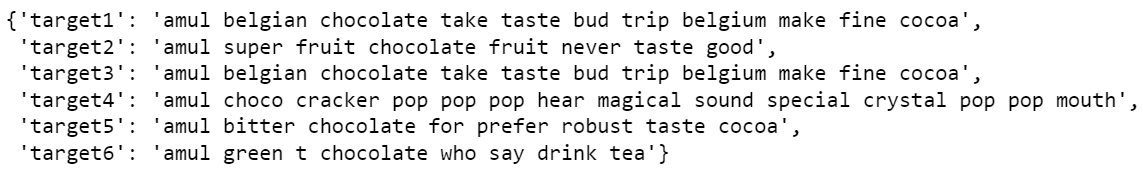


Fig: Final preprocessed dictionary

* **Jaccard-Index:**The Jaccard index or Jaccard similarity coefficient compares members for two sets see which members are shared and which are distinct. It’s a measure of similarity for the two sets of data, with a range from 0% to 100%. The higher the percentage, the more similar the two populations. [1].

**Formula : J(X,Y) = |X∩Y| / |X∪Y|**

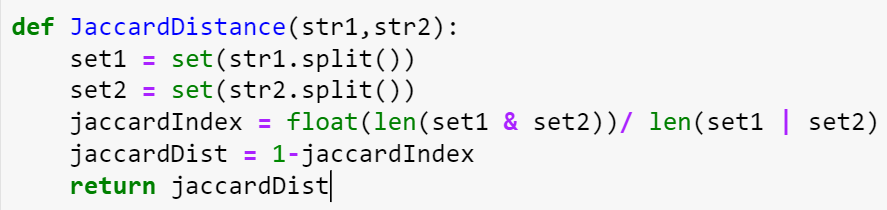
* **Jaccard-Distance:** A similar statistic, the Jaccard distance, is a measure of how dissimilar two sets are. It is the complement of the Jaccard index and can be found by subtracting the Jaccard Index from 1[1].

**Formula : D(X,Y) = 1 – J(X,Y)**

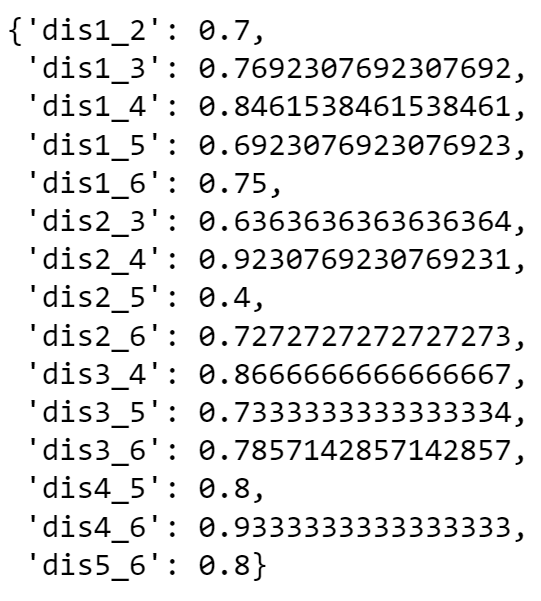
* **Triangle Inequality:** The sum of any two sides of a triangle is greater than or equal to the third side; in symbols, a + b ≥ c[2].

**Formula: ||x|| + ||y|| ≥ ||x + y||.**

* Modified function to calculate Jaccard-Distance.

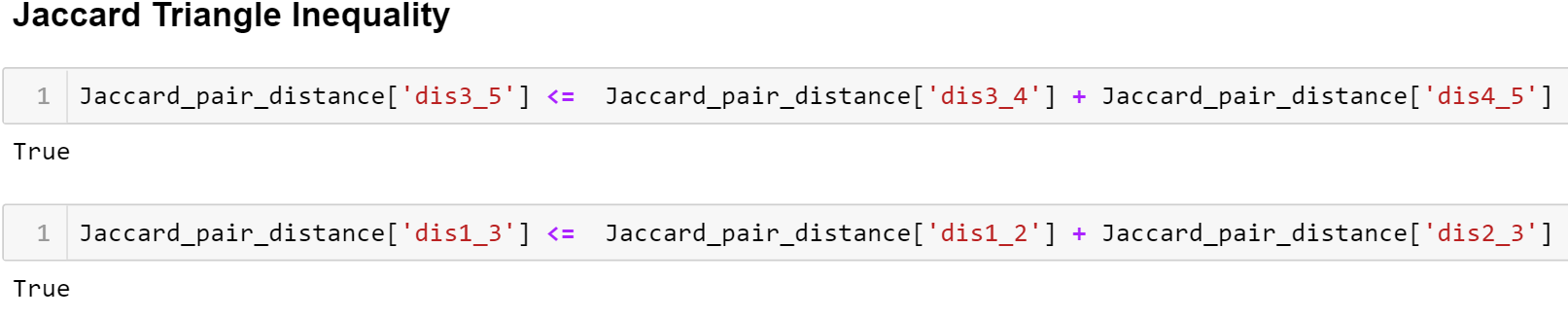


After calculating pairwise distance, following is the result.



Here ***dis1\_2,*** show distance between target1 and target2. Similarly rest all distance are calculated above.

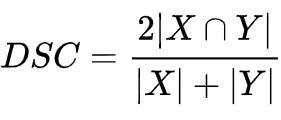
* Jaccard Distance hold triangle inequality property for each pairs. The code snippet below shows calculation for few of the pairs.



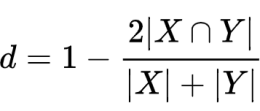
Above we are calculating jaccard-distance between pairs. It can be clearly seen that the pair satisfies triangle inequality theorem.

**b. Now implement the difference function for the Dice Coefficient and show that the property of triangle inequality may not hold for this measure.**

**Ans: Dice Coefficient:** |X| and |Y| are the cardinalities of the two sets (i.e. the number of elements in each set). The Sørensen index equals twice the number of elements common to both sets divided by the sum of the number of elements in each set.

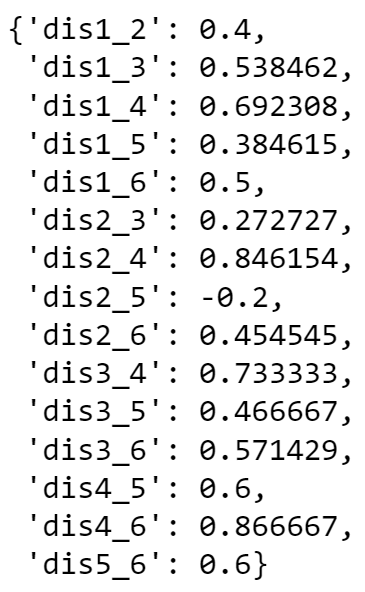


Dice coefficient distance is calculate by,

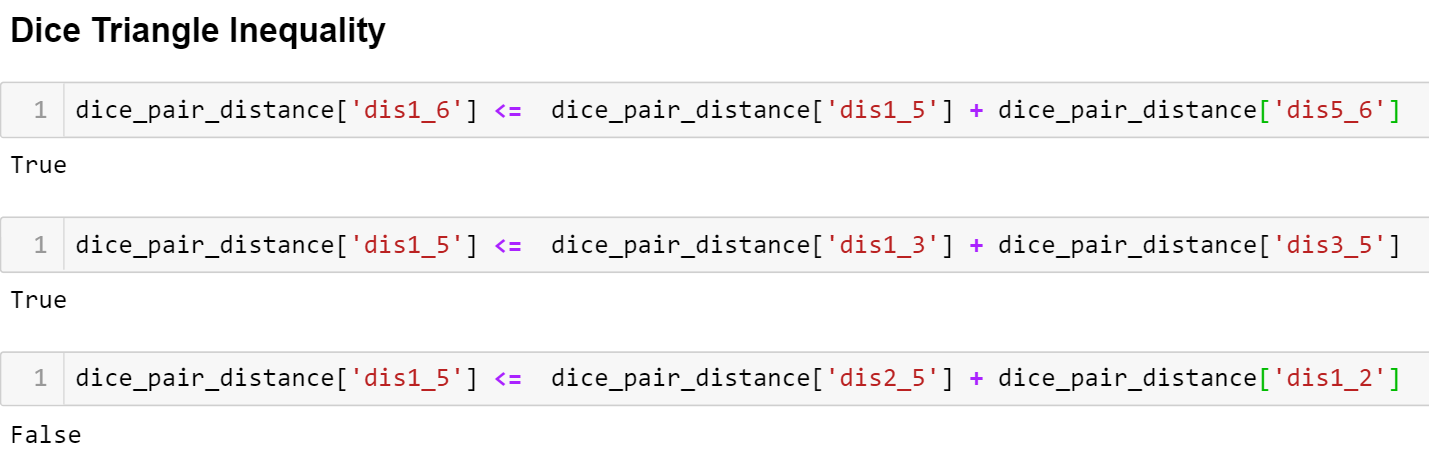


But it is not a proper distance metric as it does not possess the property of triangle inequality.

I have calculated pairwise Dice coefficient distance, following is the result.



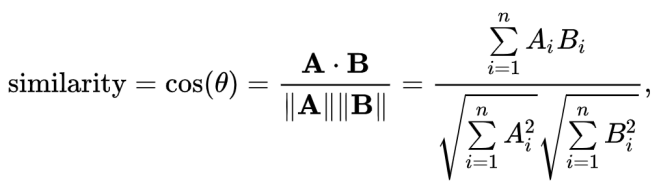
Triangle inequality doesn’t not possess by Dice distance and it can be proved by looking at below output.



**Q2. Have a look at the Cosine.py program; nb you may need to install the packages its imports.**

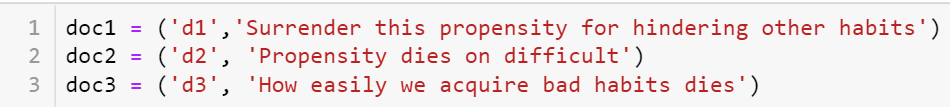
**Ans: Cosine Similarity:** Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity[3].

The Formula is given by:



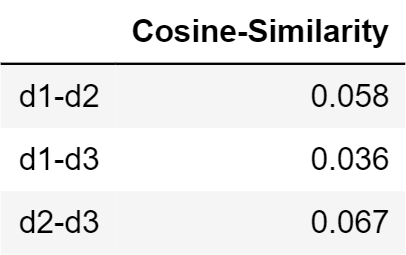
**a. Find 3 short documents about which you might want to know their similarity. Produce 5 variants on one of the documents and see how the cosine similarity changes.**

Ans: three short docs for calculating cosine similarity.

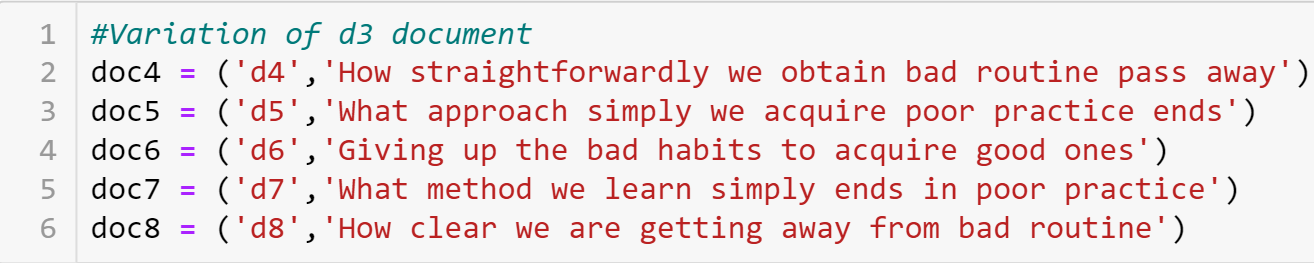


Now pairwise similarities are calculated from ***get\_cosine*** method from cosine.py file. For eg. cosine similarity between doc1 and doc2 is calculated then doc1 and doc3 and so on.

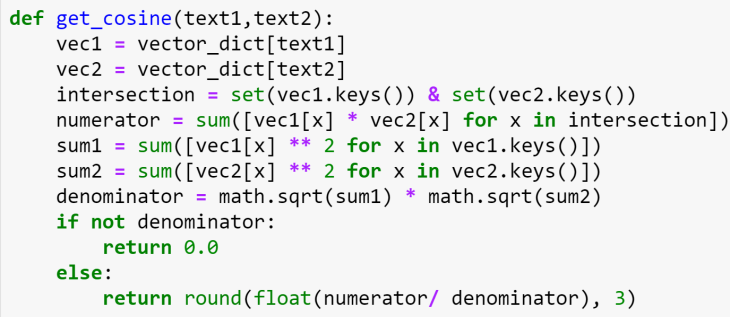
Below is the output after calculating each pair cosine value.



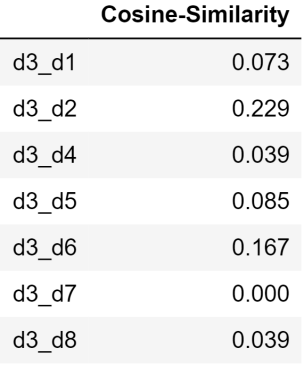
Variants of d3 doc are:



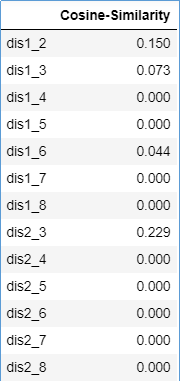
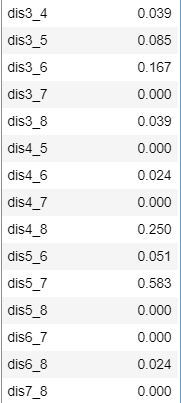
Calculating cosine distance of doc3 from doc4 to doc8. Below is the python function that is used to calculate the cosine similiarity and the output produce after pairwise comparison.



**Output:**



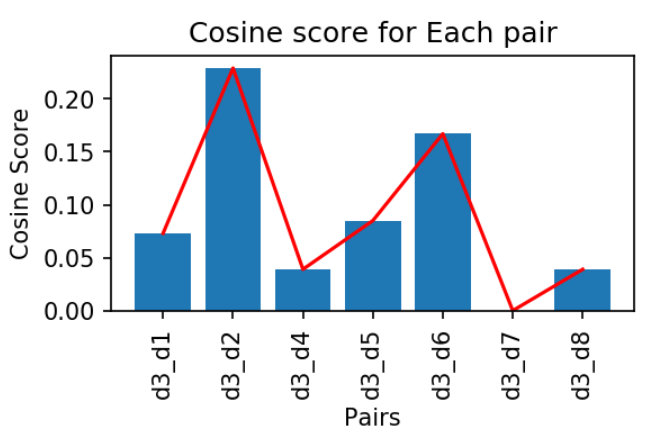
All pairwise distance from each document:

**b. Plot the similarity differences on a graph showing their cosine similarity score. Verify that your intuitions about what makes the differing docs less similar does indeed lead to scores that are less similar.**

**Ans:** The graph shows the cosine similarity score of document 3 with all other documents. The graph shows clearly that the documents are similar when the cosine score is high.

When the cosine angle between documents is less more is the similarity hence the cosine score. Hence the intuition was appropriate that when the score is less, similarity is also low.



**c. Find a python package that computes cosine similarity and euclidean distance. Use it process the data you have already. Do the answers for Cosine Similarity correspond? What do the Euclidean Distance scores look like relative to the Cosine ones?**

**Ans:** Package for cosine similarity: sklearn.metrics.pairwise.cosine\_similarity

Package for euclidean distance: sklearn.metrics.pairwise.cosine\_similarity

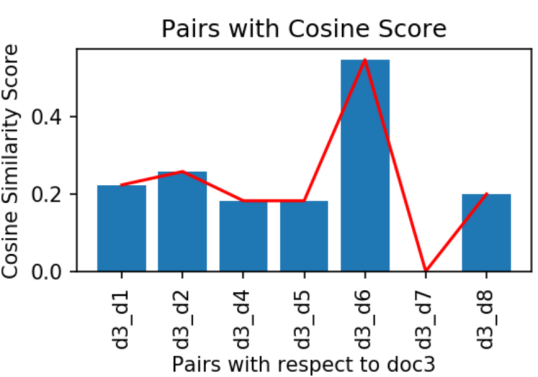
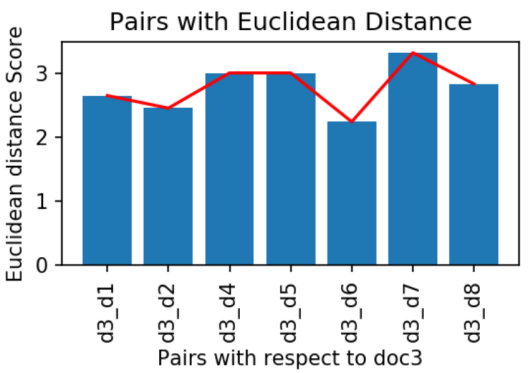
 

Fig1: Plot for Cosine Score Fig2: Plot for Euclidean Distance

Cosine similarity of doc3 with all other documents are calculated using library specified above. It internally calculates Term frequencies (TF) for most occurring terms using fit\_transform inside CountVectorizer and then using cosine similarity function. Calculating the cosine manually and through inbuilt libraries are related but values are changing because of the tf-idf values differ when calculated manually and via CountVectorizer. Fig 1 shows the Cosine Score.

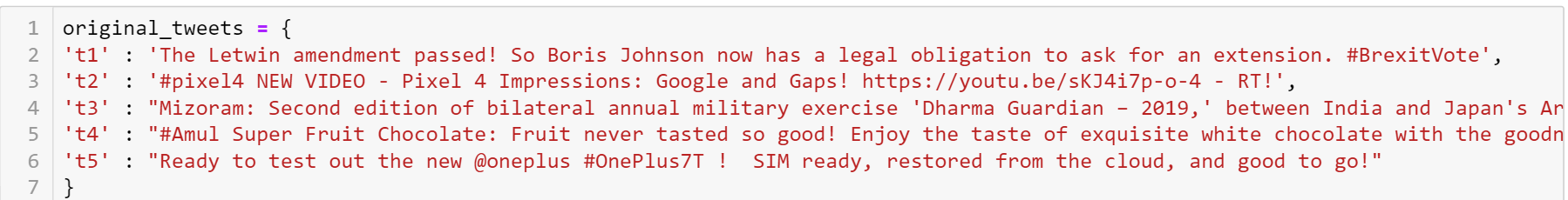
Moreover Euclidean distance is opposite to the cosine similarity. It measure the length between two sample points. It considers the magnitude whereas cosine considers the angle between the vectors.

Hence we use cosine similarity when the magnitude is not of much importance. Fig 2 shows the Euclidean distance.

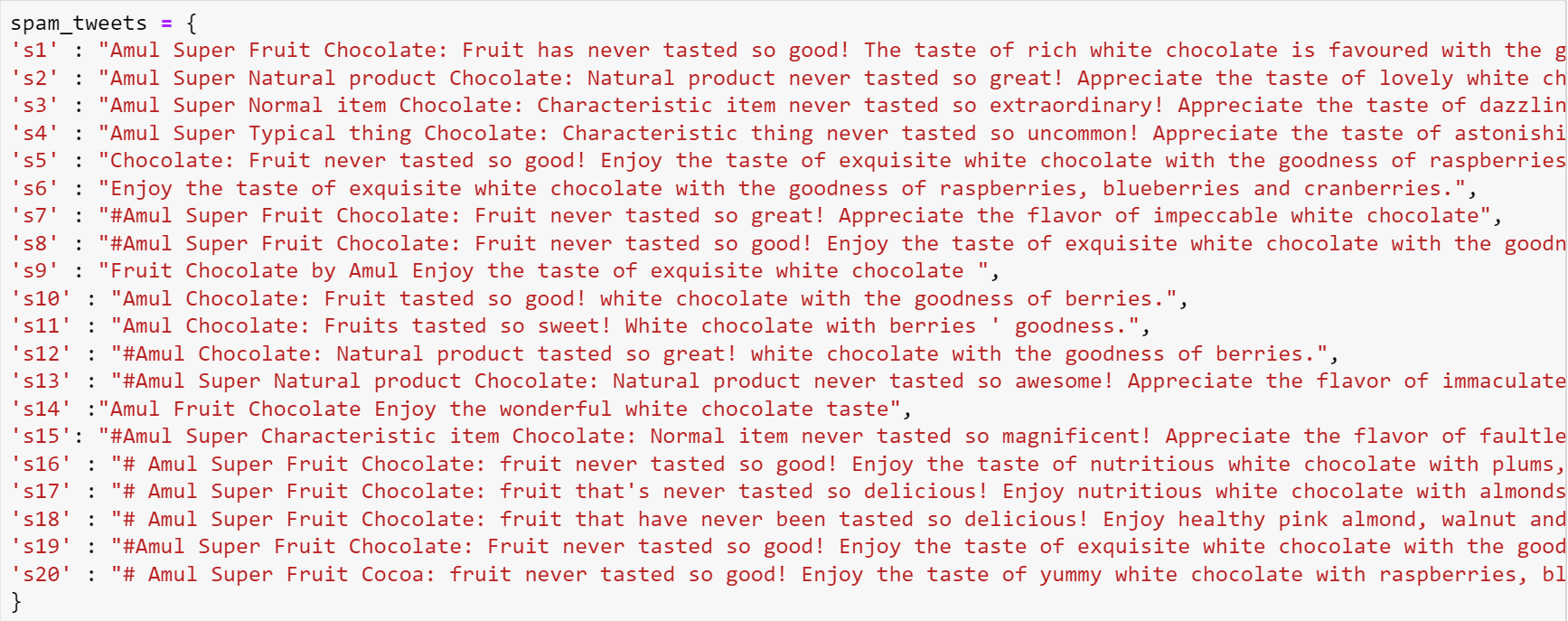
**Q3. Create or find 5 “normal” tweets from Twitter. Now take one of these tweets and sysmatically generate 20 SPAM tweets from it; using the typical techniques of spammers.**

**Now, perform comparisons between these 20 SPAM tweets each of the 5 Normal Tweets. Plot their edit-distance scores in a graph and colour code to show how the SPAM v Normal ones. Are the SPAM tweets obvious, if not why?**

**Ans:** Tweets:



Based on tweet4 20 spam tweets are created.



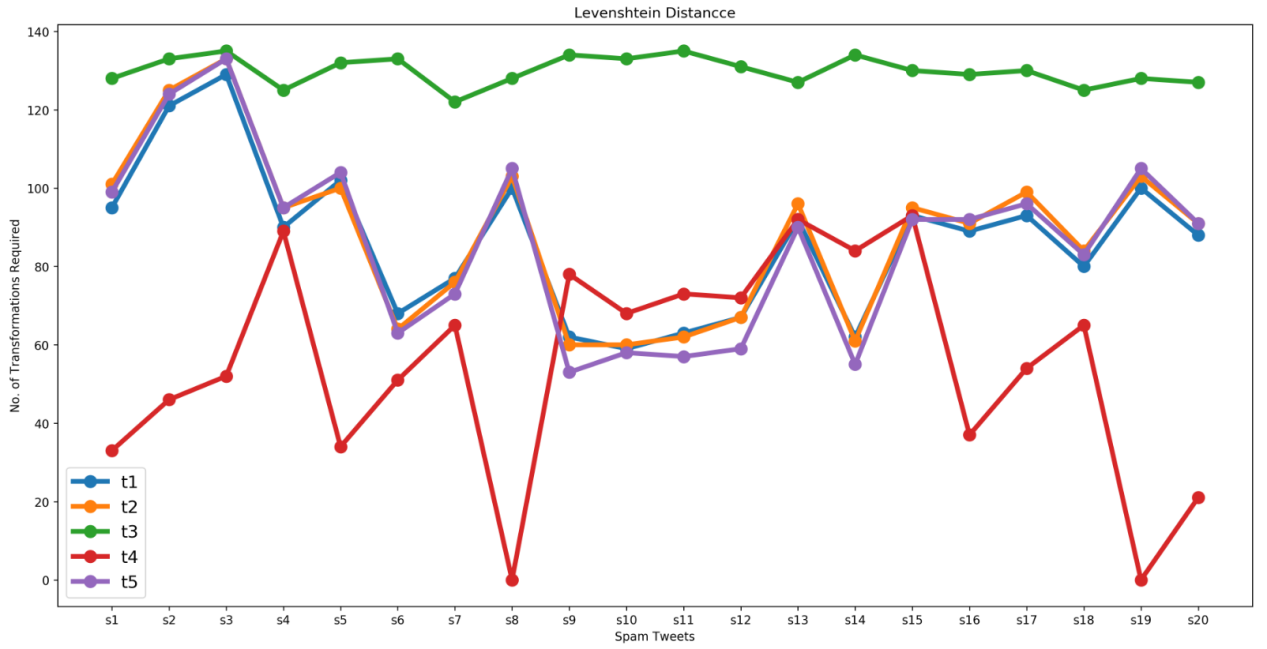
All the normal and spam tweet are preprocessed using usual procedure as mentioned in question 1 above.

**Levenshtein Distance:** It is a string metric for measuring the difference between two sequences.

The Levenshtein distance is calculated between normal and spam tweets and the result is shown below.

The plot for Levenshtein distance between normal and spam tweet is shown below.



Looking at the graph above it can be clearly seen that spam tweets are related to normal tweet **t4** as it shows it require minimum number of transformation to achieve maximum similarity measure.

**References:**

1. S. Glen, “Jaccard Index / Similarity Coefficient”, 2, December 2016 [Online]. Available: <https://www.statisticshowto.datasciencecentral.com/jaccard-index/> . [Accessed Oct. 19, 2019]
2. W. L. Hosch, “Triangle inequality”, 18, August 2009 [Online]. Available: <https://www.britannica.com/science/triangle-inequality>. [Accessed Oct. 19, 2019]
3. S. Prabhakaran, “Cosine Similarity – Understanding the math and how it works (with python codes)”, 16, May 2019[Online]. Available: <https://www.machinelearningplus.com/nlp/cosine-similarity/>. [Accessed Oct. 19, 2019]