Computing Social Score of Web Artifacts

Soumyajit Ganguly Simran Kedia
Jaspreet Kaur J N Venkatesh

Lakshminarayanan Srinivasan
Prof. Vasudeva Varma
IIIT Hyderabad

Abstract

Different social media sites have different entities, like *Facebook* has posts, *Twitter* has tweets, *Youtube* has videos, *Pinterest* has pints, etc, etc. Each of these sites have their own metrics for calculating the social score of artifacts posted in their respective sites. In this project, we propose an approach which computes a single aggregate metric by taking into account the different metrics from different social media sites.

1 Motivation

We want to compute a score of an artifact that reflects the popularity of an artifact across different social media sites and not just limited to any particular site. The computed score, spanning different media sites is a very useful metric, as it tells the popularity of an artifact across the entire social media. Possible use-cases can be:

- The Publicity Team of a company may want to know the popularity score of its advertisement video across the social media and see how people are reacting to it
- Any content creator company which creates and shares its content on different social media sites, will get to know the popularity score and get feedback about the content.

2 Introduction

Web artifacts are items like news articles, blog posts, videos, or any other URL. Each of these sites have different metrics to calculate popularity score of artifacts, for example, for *Facebook*, it would be number of likes, shares and comments; for *Twitter* it would be number of favorites, re-tweets, comments, etc, etc.

As, *Facebook* and *Twitter* are the leading social media sites and is generally the first choice for content-creators / ad agencies / freelancer / etc., to share their content, we would be focusing on computing the social score for artifacts shared on *Facebook* and *Twitter* only.

But, there are two main challenges to be tackled, in computing the score:

- An artifact could have been shared by multiple people and on multiple social sites. So, we have to aggregate artifacts from multiple sources and compute a combined score obtained for each of the artifacts.

- There will be many posts and tweets where people discuss/talk about an artifact, but they do not always quote the artifacts. Ideally, these tweets/posts should also contribute to the popularity of the artifact. But, matching these tweets/posts to an artifact is quite a bit of challenge.

For example, the *Mauka Mauka* Ad videos became very famous on social media, at least in Indian sub-continent, during the ICC Cricket World Cup and many people shared it on multiple sites and people talked about it for length, without necessarily quoting the video link in every post/tweet.

If we were to compute the popularity of this *Mauka Mauka* ad, we would have to consider both the tweets/posts which have quoted the link of the artifact as well as those tweets/posts which talk about the artifact.

3 Problem Statement

So, now that we have sufficient background information and challenges about the topic, would now formally define the problem statement.

Given datasets of Facebook and Twitter crawled data, we would compute the popularity score of all the artifacts present in the datasets. We will compute this score by aggregating not only the tweets/posts which have quoted the artifacts but also the tweets/posts which talk about the artifact, and assign a combined score obtained for each of the artifacts.

Throughout this report, we use the terms document and artifact interchangeably. This is because while dealing with text mining algorithms the data extracted from every artifact (website) is treated as a document.

We describe the algorithms used in section 2. and detailed procedure of how it is done in section 3. The final results are discussed in section 4. and we conclude in section 5.

4 Approach

To start with, we first built a naive model, where the social score of an artifact shared on *Facebook* and *Twitter*, would just be the weighted sum total of number of likes, shares and comments of *Facebook* posts and number of shares and favorites of *Twitter* tweets containing that artifact.

But, as we have discussed, this is not a correct measure of the popularity of an artifact. As, there may be tweets/posts talking about an artifact and may not always quote the artifact.

So, we developed a more sophisticated model, where we followed these steps for computing the popularity score as described in the problem statement:

- **Gathering data / Crawling** : We crawled the *Facebook* and *Twitter* pages for gathering data. For crawling facebook data we used the python facebook-sdk which uses GraphAPI [13] for dowloading the data in json format. For twitter we used a java library twitter4j [9] which downloads the data in json format.

- **Pre-Processing data** : For extracting the text from artifacts, we used html2text tool [12]. The text data as obtained from the html2text tool output is tokenized. All punctuation marks and special characters are removed. For this, we used the help of python NLTK tokenizer[11] which does a good job as it is trained on a large corpus of English data.

  After tokenizing, we performed stop-word removal and stemming [11] which are two usual steps almost always performed for text data processing. The stop-word removal step removes a lot of noise and also reduces the dimensions of text. Stemmming helps mitigate effects of tense and other grammatical correctness which is really not a concern for text processing tasks.

- **Algorithm** We tried 3 different algorithms for our task. The algorithms are as follows:

  1. ***kNN Classifier***
     - Create histograms for every document using count vectorizer.
     - Normalize the histograms by constructing tf-idf
     - We now have vector representations for each document (text crawled from artifact)
     - The training data would be the documents containing text of all the artifacts.
     - Now, train the k-NN classifier in an unsupervised model using all the tf-idf vectors with default parameters.
     - Then consider the non-artifact posts.
     - Convert the text of non-artifacts to respective feature representations using the same vocabulary as in training.
     - As a final step, compute the top-10 closest neighbors(artifacts) for each non-artifact and computed the respective similarity scores.
     - Now, given a non-artifact, we consider, that this non-artifact may be talking about the 10 artifacts that we get as results of kNN.

- So, for every artifact in the results of kNN (for each non-artifact), we update its likes, shares, and comments count as follows:

  $$NewCount = OldCount + NormalizedSimilarityScore * CountOfNonArtifact$$

  Here OldCount refers to the original naive count of the artifacts. The count refers to the like/share/comment count of an artifact.

2. ***Indexing***

   - Take the text of the artifacts and put them in a directory for indexing, such that each file contains the text of each artifact.
   - Indexing is done for the various files with artifacts and the indexed structures are stored in a separate directory.
   - All non artifacts are then queried against the indexed files and 10 most closest documents per query are retrieved.
   - So, for every artifact/document in the results of Indexing (for each non-artifact), we update its likes, shares, and comments count as follows:

     $$NewCount = OldCount + NormalizedSimilarityScore * CountOfNonArtifact$$

     Here OldCount refers to the original naive count of the artifacts. The count refers to the like/share/comment count of an artifact.

3. ***k-Means Clustering***

   - Do the pre-processing steps as carried out as in kNN and thus will have tf-idf vector representations for each document.
   - Now, take all vectors and try to cluster them using k-means algorithm.
   - After the clustering is done, analyze the results of each cluster.
   - All the non-artifacts in each cluster may be considered as, they are related to the artifacts in that cluster.
   - So, these non-artifacts in each cluster should contribute to the overall score of an artifact.
   - The new count (of like/share/comment) of an artifact can be calculated as:

     $$NewLikeCount = OldLikeCount + \sum LikeCountOfNonArtifact$$

     $$NewShareCount = OldShareCount + \sum ShareCountOfNonArtifact$$

     $$NewComntCount = OldComntCount + \sum ComntCountOfNonArtifact$$

   - We apply this formula for each and every artifact in the cluster.

- **Database / Scoring** : The results obtained from kMeans/Indexing/Clustering of non-artifact tweets/posts is used to calculate the overall likes,comment,share count of an artifact. Only a fraction (depending on the similarity score) of the like/share/comment count of non-artifacts is added to the like/share/comment count of the artifacts. Once we get the overall likes,comments and share count of an artifact, its score is computed by the formula:

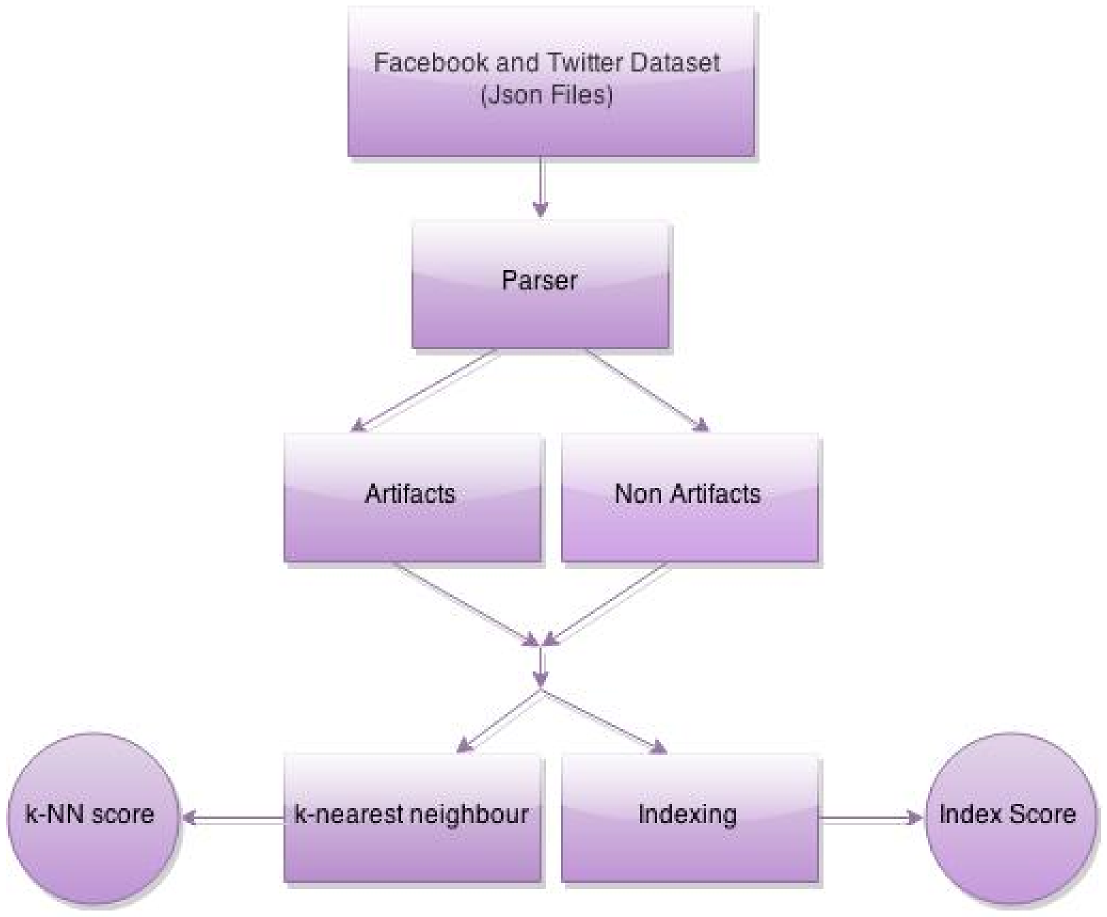
  $$FacebookScore = x * no\_of\_likes + y * no\_of\_comments + z * no\_of\_shares$$

  $$TwitterScore = x * no\_of\_favorites + y * no\_of\_comments + z * no\_of\_retweets$$

  x,y,z refers to the weights we want to give to likes/comments/shares.In our experiments we kept x=1, y=2 and z=5, thus giving more weightage to shares.

5 Experiments

Figure 1. System Architecture



For preparing dataset, we crawled all the Facebook and Twitter pages related to the *ICC Cricket World Cup.* We crawled the data from 1st January 2015 to 1st April 2015 from Facebook and Twitter, and all our seed URLs were cricket related. So, our entire artifacts and non-artifacts is revolved around cricket and world cup and other minor tournaments. We got around 8770 unique artifacts from the crawled data of Facebook and Twitter.

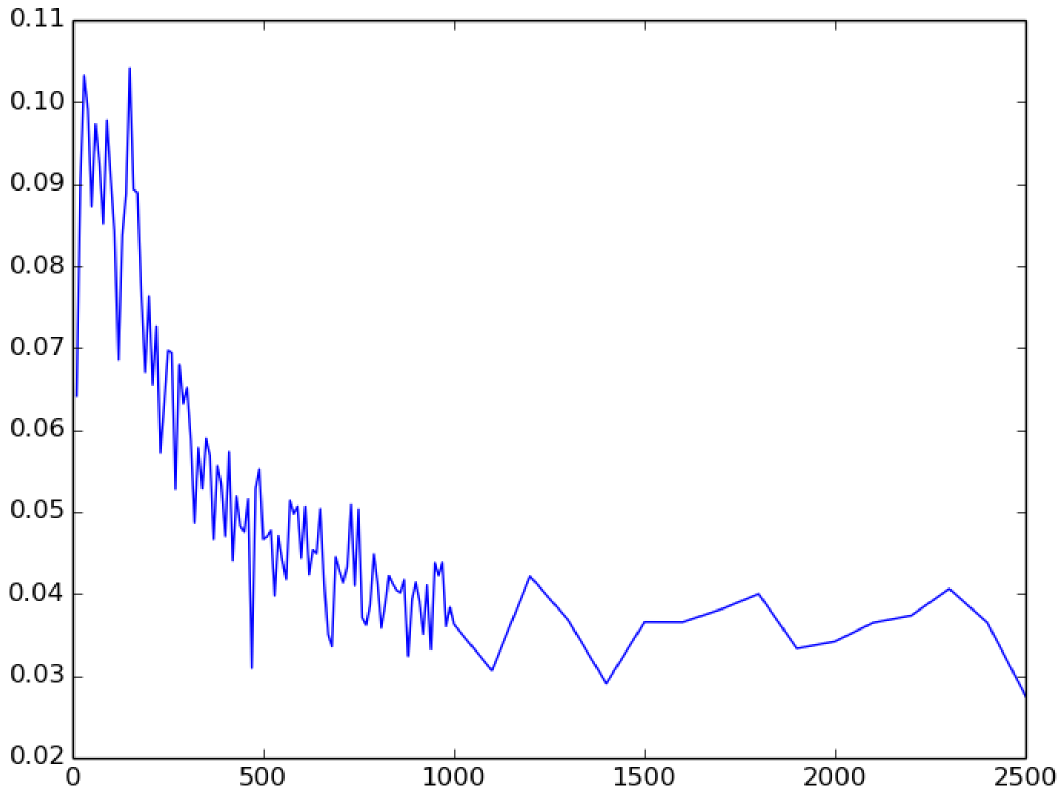
kNN: We used Scikit-learn [3], a Python library for implementing the kNN algorithm. Then we performed each step as mentioned in the Algorithm and the results we got are shown in the next section.

Indexing: We used Lucene for implementing our indexing approach. Lucene uses several optimizations for indexing the data which is helpful for quickly finding similar documents. [6] Lucene uses a weighted combination of Okapi-25 and binary retrieval model for the retrieval [8]. The results we got are shown in next section.

k-Means: We used Scikit-learn [3], a Python library for implementing the k-means algorithm. Choosing a good value of 'k' was a difficult task. We tried with k as 100 and went up till k is 2500 and recorded the silhouette score which measures the inter-class and intra-class distances. But we found out that a marginally better value of 'k' was about 180. Refer Figure 2. This was not a satisfactory result and the effective clusters so formed using 'k' as 180 were not of good quality (manually checked).

We tried to find out the reason for the poor results k-means was giving us. The naive k-means algorithm uses euclidean distance metric but for text documents, cosine similarity is always a better measure as it is immune to the l2 norm of the vectors. The centroid computation of even separate metrics in k-means lie on the basis of central moments which comes from euclidean space geometry. This resulted in inferior cluster formations. So, we decided not to go ahead with this approach and thus do not show its results in the next section.

Figure 2. Silhoute Score v/s Number of Clusters



6 Results

As mentioned before, the dataset we crawled is about *ICC Cricket World Cup*. Ideally the most popular video should be Australia winning the World Cup. As a matter of fact, in both our methods, this artifact came out to be the top scorer. This article was from the official site of ICC.

The second spot differed in the kNN results and Lucene results, but, for both the approaches, top results that differed from each other were within short distances (in ranks) from one another. A post about Srilankan cricketers retiring also got a high score from both and was within top 5 artifacts. This is also expected, because legends like Sanath Jayasuriya and Kumar Sangakkara were retiring and many people shared this artifact and talked about it.

Before seeing the results, we thought that the famous *"Mauka Mauka"* ad from Star India would also be one of the high scored artifact. But it wasn't the case. As a matter of fact, it was not in top 20 from either methods. This may be due to the reason that although the series of ads were very common and talked about in India, for the rest of the cricket playing nations it was not so.

As output, we will be providing two files from two different approaches we tried - kNN and Indexing. The format will be (artifact, social score) for each of the artifacts for both the methods.

7 Future Work:

- Presently, we are crawling the past data and then applying our algorithm. We can extend this to make it more dynamic, as every second, hundreds of new tweets and posts come up on Twitter and Facebook respectively.
- Better techniques for data crawling could be used. Presently, there are some links, from which we could not crawl the content, due to the error - ”need cookies to access this website”
- We will incorporate a decay factor while calculating the social score to ensure that the most relevant information influences the computed popularity to an appropriate degree because the popularity of an artifact may fluctuate over time.
- While computing the score of an artifact, if any verified user tweets or shares an artifact, then we could give more importance/weightage to these posts while calculating the overall score of the web artifact.

8 Conclusions:

We tested this project of ours only for cricket related datasets, but this same thing can be very well used when trying to see whole year activity of political agenda or viral videos or important news outbreaks, etc.

9 References

1. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze. Introduction to Information Retrieval, 2008
2. Charu C. Aggarwal, ChengXiang Zhai. Mining Text Data, *Springer*, 2012
3. Pedregosa et al. Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research 12, pp. 2825-2830*, 2011
4. Bentley, J.L. Multidimensional binary search trees used for associative searching, *Communications of the ACM*, 1975

5. Omohundro S.M, Five ball-tree construction algorithms, *International Computer Science Institute Technical Report*, 1989

6. Bawa, M., Condie, T., Ganesan, P., LSH Forest: Self-Tuning Indexes for Similarity Search, *Proceedings of the 14th international conference on World Wide Web*, 2005

7. Jiangong Zhang, Xiaohui Long, Torsten Suel, Performance of Compressed Inverted List Caching in Search Engines, *17th International Conference on World Wide Web*, 2008

8. Gianni Amati and Cornelis Joost Van Rijsbergen, Probabilistic models of information retrieval based on measuring the divergence from randomness, *ACM Transactions on Information Systems*, 2002

9. Yusuke Yamamoto, Java library for the Twitter API, *http://www.twitter4j.org/*, 2007

10. Massimo Di Pierro, web2py for Scientific Applications, *Computing in Science and Engineering, vol.13, no. 2, pp. 64-69*, 2011

11. Steven Bird, Ewan Klein, Edward Loper, Natural Language Processing with Python, 2009

12. *https://pypi.python.org/pypi/html2text*

13. *https://github.com/pythonforfacebook/facebook-sdk*

14. Alan Ward, Computing popularity based upon subscriptions. *US 11/366,321*, filed Mar 1 2006, and issued Sep 6 2007.