Meme Tracker

Social Network Generation and Ranking

Eshwaran Vijaya Kumar
Dept. of Electrical & Computer Engineering
The University of Texas at Austin
eshwaran@utexas.edu

Saral Jain
Dept. of Computer Science
The University of Texas at Austin
saral@cs.utexas.edu

Prateek Maheshwari
Dept. of Computer Science
The University of Texas at Austin
prateekm@utexas.edu

Categories and Subject Descriptors
H.4 [**Information Systems Applications**]: Machine Learning, Text Mining

General Terms
Machine Learning, Text Mining, Mapreduce

1. INTRODUCTION

This document serves two purposes: We describe and illustrate our project, explain progress made and talk about challenges that we are currently facing and steps that we plan to take to overcome those. We also illustrate a plan of action to tie up the several components of the project together.

A meme is a short phrase which originates, travels and mutates in a social network. The meme is a potentially undiscovered path through which information flow takes place in a social network. We would like to automatically extract short phrases which are present in some possibly mutated form in a group of blog posts, group similar phrases together to identify nodes (blog posts) that share edges. This meme-graph is then evaluated as follows: We compare the performance of the meme-graph as a ranking mechanism when compared to a baseline social-graph, the hyperlink graph exploited successfully by Google.

2. MEME-GRAPH GENERATION

2.1 Extraction of a Meme

The first part of the problem is that of automatically identifying and tracking memes from a corpus of documents that are potentially linked to each other by memes. There are several ways to define a meme [?], [?] which impose fundamental assumptions on the design of the entire system.

Kolak and Schilt [?] describe a scalable technique for generating quotations that are shared between pairs of documents that we employ for the meme generation portion. We describe this in the next section and talk about implementation challenges that we face.

2.1.1 Sequence Generation Overview

Given a "clean" corpus of documents (blog posts), we can generate key-value pairs where a key is a short contiguous sequence of characters and value is a pair of documents that contains that key as follows: We first parse each blog posts and generate shingles composed of a few tokens where every token is a space delimited collection of characters. These shingles are used to build an inverted index where the keys are shingles and value is a bucket list. The bucket list is a data structure that is an array of buckets where each bucket contains a unique identifier for a blog post and a position marker which identifies the start of the shingle in the document. The next phase is to merge shingles that are contiguous to each other in a pair of document. This is done as follows: We walk through each document, generate all the shingles that exist in that document. We create a set of sequences as follows: We walk through the shingles in the source document, check if

The number of tokens is not fixed to a constant in our system but so that we can tune it empirically.

Leskovec et. al.[?] make two fundamental assumptions which we feel impose too much of a restriction on the latent network generated: 1) They assume that every meme is demarcated by quotation marks. While, this assumption might be true for a subset of the memes, it would result in a system that fails to capture a large number of memes that don't possess this characteristic. 2) They assume that the evolution of a meme over time can be tracked by the increase in length of the meme. It is not clear why this assumption should be true in general, or why the inverse shouldn't happen. Since we have temporal information, we would like a system that takes that into account. Kolak and Schilit [?] approach a similar problem which can be applied to solve the problem of meme extraction: They mine quotations from books and use them as a way of generating a cross reference across books. However, we feel that their technique for string matching can be improved upon by incorporating more sophisticated approaches for calculating string similarity, like vector space models or language models.

2.2 Direct String Similarity

The approach taken by Kolak and Schilit, i.e., generating and extending shingles and then grouping the final results, has a few drawbacks. For one, they set the minimum size of shingles to 8 to reduce the amount of intermediate output. A long shingle length would require sentences containing meme to have a significant and exact overlap, an assumption that is not necessarily true in practice. For example, the following two sentences have the same information content, but would not be detected by the shingles approach for a shingle length greater than 5 (corresponding to the ngram "at the Intel Developer Forum"). One way to overcome this limitation would be to use a smaller minimum size for shingles. However, setting the shingle size lower would increase the amount of intermediate output and the size of each bucket list, and will also increase the amount of time spent extending the shingles for each document in the next phase. Also, in the shingling approach, we generate the shingles irrespective of sentence boundaries, which generates many redundant shingles. (Is this true?)

```
The processors were announced in San Jose at the In-
tel Developer Forum.

The new processor was unveiled at the Intel Developer
Forum 2003 in San Jose, Calif.
```

Another problem is that if the different variations of the meme have been paraphrased, the shingling approach would limit the detectable meme to the longest sequence that two sentences share. Furthermore, if the memes are generated by following a shingling approach, we need to group the memes that share common subsequences into their common "superstring" by grouping the shingles based on the words they share. This problem is equivalent to finding a hamiltonian path in a graph of the meme strings in which the strings are the nodes and all memes that share a substring are directionally connected. The problem of finding Hamiltonian paths in a graph is known to be NP-hard(or complete?) and the solution will not necessarily give accurate results since the meme strings might have an ngram overlap without being originally related.

To avoid these problems we propose an alternate approach to generating and grouping the memes based on a pairwise string similarity computation. For thise approach, we consider a sentence to be the atomic unit that conveys information related to a meme. The idea behind the process is to shortlist the strings from the document collection that might be related to each other and compute one or more syntactic/semantic string similarity measures for those pairs of strings. If the score is above a certain threshold (to be determined experimentally), they will be considered to be the same meme. The process is described below in more detail.

2.2.1 *Preprocessing for Shortlisting Pairs*

In the pairwise string similarity approach described above, we need to shortlist the strings that might potentially be the same meme and compute their similarity. Given enough computing power and time, we could compute the similarity of all the strings in the corpus with each other. However, this is not practically feasible given the large number of strings, and hence we shortlist the strings we will be computing the similarity scores for. To shortlist the strings, we make an assumption that similar strings share at least one ngram (of a size to be determined experimentally) and vice versa, that each pair of string that shared the same ngram is a candidate for similarity computation. We construct an inverted index from ngrams to a postings list containing strings that contain that ngram from the document collection. If we take a cross product of the postings list for each ngram with itself and keep unique string pairs, we get a list of the string pairs that are candidates for similarity computations.

2.2.2 *Experimental Results*

We now present the experimental results from an analysis of the effect of ngram size used for shortlisting pairs on the efficiency and effectiveness of retrieval.

2.2.3 *Grouping*

The next stage is to identify all variants of a single meme and identify the latent social network whose nodes are blog posts and edges are the common meme shared between them. As a first pass, we can try to handle typographical errors, sentence breaks and short changes in the sentence structure using a heuristic that was utilized in [?]. For any given node (post), find all nodes that share a portion or all of the meme; Use this information to discover groups of memes that should ideally be one meme. This allows us to generate the meme-graph which can also be directed based on temporal information.

2.3 More Sophisticated Extraction Methods

We feel that the basic string matching algorithm can be improved upon by a number of techniques. One possible approach is to use vector space models (e.g., cosine similarity) to calculate similarity on individual sentences; this will tell us if two sentences are similar without a substring matching. The motivation is to find memes that might be different by simple transpositions of phrases or words, or differ by a few words at most.

Language models provide a statistically sound framework for text modeling and have been popularly used in speech recognition, topic modeling, machine translation etc [?]. As another option for calculating string similarity, given the language model of sentence 1, we can try to calculate how likely it is to generate sentence 2, and consider the sentences similar if the generation probability is high.

Another potential approach would be to perform direct text clustering on sentences from a collection of documents to find a set of similar sentences and then construct a shingle table using the documents the sentences belong to.

We will explore these intuitions further in conjunction with insights obtained from existing research in string similarity[?], quotation mining and other related areas such as plagiarism detection [?].

3. GRAPH CLUSTERING

Once the latent meme-graph has been generated, we first collapse all nodes for posts from the same blog into a single node for that blog. After this step we have reduced the social graph of posts to a social graph of blogs. We then try to analyze the structure of the graph(s) by clustering the nodes using either [?] or [?]. This will help us identify cliques of blogs that have a high information flow between them due to frequent exchanges of memes, and therefore often participate in discussions about the same issues. The algorithm discussed in [?] is a multi-stage algorithm where each node is initially put into its own community and we iteratively merge communities until the objective function defined for the graph attains its local optimum value. Although we have experimentally tested the scalability of the algorithm in another context using large social networks, it would be interesting to try to develop an equivalent MapReduce variant of the algorithm. Alternatively, we could also utilize the libScotch library [?] to perform graph clustering in a distributed (albeit non-MapReduce) fashion.

4. PROJECT SCHEDULE

We give a brief overview of the organization of the project. We invite the interested reader to refer to the pivotal tracker for the project for more details.

Data Preprocessing The data will have to be initially pre-processed using a HTML parser and be converted to JSON. We have sequential code performing this task right now on a subset of the data. We would like to extend this code and construct a MapReduce version using Python Streaming.

Basic String Matching Algorithm We would like to implement the basic string matching algorithm outlined in [?]. This stage will be split into two components: We would first like to develop a sequential version of the algorithm and then extend it further to develop a MapReduce variant. This stage will be used to identify the latent social graph that is generated by exact string matches. Once this is complete, we can optimize this further to perform preliminary groupings in the fashion developed in [?]. Another issue we have not sorted out at this stage is how much of the grouping can be done in a distributed fashion.

Sophisticated String Matching There are several interesting threads that we plan to follow in this phase. We could improve upon the grouping in the previous stage by using a Vector Space Model to try to improve the grouping. This should be easily doable in a MapReduce fashion. Some other interesting approaches would be to utilize language models to improve upon the basic string matching that is being done initially. This phase will also involve exploring MapReduce models to build these models.

Graph Clustering As a first step, we have sequential code available from the authors of [?]. Depending on the size of the social networks generated, we would have to explore whether or not, we would have to write a MapReduce variant of this algorithm in order to perform clustering. Another issue with this approach is trying to optimally merge graphs generated by different memes in order to construct one single meme-graph. Depending on which of these tasks proves to be harder, we could switch to using a multi-graph approach as outlined in [?].