# Meme Tracking in Scale with MapReduce

Hohyon Ryu
School of Information
University of Texas at Austin
hohyon@utexas.edu

Jae Hyeon Bae
Computer Science
University of Texas at Austin
metacret@gmail.com

Nicholas Woodward
School of Information
University of Texas at Austin
woodward.nicholas@gmail.com

## ABSTRACT

This article is to report our project progress for Fall 2011 Data-Intensive Computing for Text Analysis class. We have completed preprocessing of the Blogs08 collection by removing all HTML tags, boilerplate text, non-English documents, and duplicates. After the preprocessing, aproximately 4 million popular common phrases were extracted using an enhanced version of the technique suggested by [2]. The extracted phrases are clustered in 2 million meme groups. The meme groups will be ranked automatically using Support Vector Machine, based on the crowdsourced labels obtained from an Amazon Mechanical Turk.

## Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia

## General Terms

MapReduce, Blogosphere

## Keywords

Keyword Extraction, Data-intensive Processing, Implicit links in Blogosphere, Meme tracking

## 1. INTRODUCTION

Web logs, also known as blogs, are a very important and interesting source of diverse information. They provide insight into the culture of the society in which they exist. People search blogs to find out how to cook, what to do, what to see, and to discover what others think. Because blogs are typically written in plain language by individuals enthusiastic for their interest, they may be more easily understood than longer, more-detailed news articles and cover a wider personal range of topics. As blogs are highly sensitive to fast-changing temporal issues, finding and tracking memes, pieces of information that travel across blog posts, we can mine variety of issues and fads in the blogosphere. In this project we will extract memes from blogs, cluster them by their similarity, and rank the meme clusters using crowdsourcing-powered machine learning.


*Data-Intensive Computing for Text Analysis* Fall 2011, School of Information, University of Texas at Austin


## 2. RELATED WORK

## 3. MEME EXTRACTION

### 3.1 Test Collection

BLOGS08[1] is a TREC[2] test collection that crawled Web Logs from Jan/14/2008 to Feb/10/2009. It consists of 3 components: feeds, permalink documents, and homepage documents. We use only permalink documents that contain 28,488,766 documents and whose uncompressed size is 1445GB. Because the collection contains many duplicates, non-English blog posts, advertisement, and Blog website frameworks, we will use NLTK and Decruft to extract English blog contents without boilerplate text.

### 3.2 Preprocessing Blogs08

Preprocessing of the Blogs08 Collection was completed using Python with NLTK and Decruft packages. Decruft extracts only meaningful content from a blog page, filtering out navigation links, advertisements, sidebar contents, and links to other sites. NLTK uses a language identification module to determine if the extracted content is written in English. Blogs08 also contains a tremendous amount of duplicate documents and documents without a significant amount of text. We set a lower-bound of 5 words and removed duplicates, also using Python. After all these preprocessing steps were completed, extracted contents of a blog post are written into a single line per document for the MapReduce meme extraction process. The number of Decruft-applied, deduplicated, and HTML stripped English blog posts is 15,097,266 (43 GB).

- English only
- Decruft
- Deduplication
- HTML Strip

[1] http://ir.dcs.gla.ac.uk/test_collections/blogs08info.html
[2] http://trec.nist.gov/

## 3.3 Meme Extraction with MapReduce

Preprocessed Blogs08 data is comprised of key-value pair where the key is document number(doc_id) and the value is the preprocessed content of the blog posts. Using Hadoop Streaming, Python MapReduce code extracts common phrases and group documents by them. Overall algorithm design follows that of [2]. However, because of a scalability issue, lower bounds and upper bounds limit the output size of the data, and other optimizations are used such as stop word ratio check and preposition trimming. The three steps of MapReduce pseudo-code for Meme Extraction are illustrated in the following pseudo codes. The first MapReduce process splits the input text into sentences and extracts trigrams and their indexes within the document. The reducer removes the common phrases(*trigrams*) that appear less than 5 times or more than 300,000 times.

**MapReduce 1:**

*Mapper 1:*

```
for (doc_id, text) in [input file]:
    sentences ← split text by sentence boundary

    for sentence in sentences:
        index ← 0
        for trigrams in sentence:
            index++
            emit (trigrams, (doc_id, index))
```

*Reducer 1:*

```
lower_bound ← 5
upper_bound ← 300000
for (trigrams, [(doc_id, index)]) in [mapper 1 output]:
    len ← count([(doc_id, index)])
    if lower_bound < len < upper_bound:
      emit (trigrams, [(doc_id, index)])
```

The second MapReduce process sorts the trigrams and indices by document to concatenate adjacent trigrams in MapReduce 3:

**MapReduce 2:**

*Mapper 2:*

```
for (trigrams, [(doc_id, index)]) in [reducer 1 output]:
    for (doc_id, index) in [(doc_id, index)]:
        emit (doc_id, (trigrams, index))
```

*Reducer 2:*

```
for (doc_id, [(trigrams, index)]) in [mapper 2 output]:
    emit (doc_id, [(trigrams, index)])
```

The third MapReduce process concatenates the adjacent trigrams within a blog post. Mapper 3 concatenates adjacent trigrams into a meme. Before emitting the memes, a function trimPrepositions trims off prepositions at the beginning and end of the meme, and a function stopwordRate checks how many of the words in the meme are stop words. If stop words constitute more than 65% of the words in the meme, the meme is ignored. In the reducer 3, phrases that are too short ($\leqq 5$) or too long ($\geqq 200$) are ignored as they are often timestamps, advertisement, or boiler plate text. The final MapReduce is illustrated as follows:

**MapReduce 3:**

*Mapper 2:*

```
for (doc_id, [(trigrams, index)]) in [reducer 2 output]:
    prev_index = 0
    meme = ""
    for (trigrams, index) in [(trigrams, index)]:
        if index − prev_index ≦ 3:
          meme += trigrams[3-(index − prev_index):]
        else:
          meme=trimPrepositions(meme)
          if stopwordRate(meme)<0.65:
            emit(meme, doc_id)
          meme=trimPrepositions(meme)
          meme.append(meme)
          meme=trigram
        prev_index = index

    # clean up for the last one
    meme = trimPrepositions (memes)
    if stopwordRate(meme)<0.65:
      emit(meme, doc_id)
```

*Reducer 3:*

```
for (meme, [(doc_id)]) in [mapper 3 output]:
    if 5 ≦ len(meme) ≦ 200:
      emit (meme, [(doc_id)])
```

The final output is key, value pairs of a meme and the list of the document ids that the meme appears. In total, 4,610,100 memes are extracted from the TREC Blogs08 collection.

# 4. MEME CLUSTERING

## 4.1 Mahout LDA

Latent Dirichlet Allocation (LDA) [1] characterize the document as the mixture of latent topics and the topic as the distribution of words. This is the most advanced topic model and interprets the document as an unordered bag of words. Each word in the document is generated by first choosing the topic from topic distribution and choosing the word from the word distribution given the topic. When we denote the word, topic, document as $w, z, D$, the topic model formulates the following word distribution in the document:

$$P(w) = \sum_{i=1}^{K} P(w|z_i)P(z_i)$$

where $K$ is the number of topic for the corpus, $P(z_i)$ is the probability that $i$th topic was sampled for $w$, and $P(w|z_i)$ is the probability of $w$ given the $i$th topic. Using this interpretation, if we find the topic and word distribution for each document to maximize the probability of entire corpus, we can group memes sharing similar sematical meaning.

Apache Mahout[3] is the open source scalable machine learning library running based on Hadoop. It has various machine learning algorithm covering from clustering, recommendation, and classification. Apache Mahout has a LDA implementation but it has several drawbacks we decided to choose for our experiment.

### 4.1.1 Defect 1: Parameter estimation

[3]http://mahout.apache.org

The first defect is it does not estimate parameters with variational Expectation Maximization (EM). In variational inference algorithm, Dirichlet parameters $\alpha, \beta$ should be estimated to maximize likelihood of $p(\mathbf{w}|\alpha, \beta)$. Since a likelihood also is not tractable for computation, variational EM method should be used, which is to maximize the lower bound of a likelihood value. But this implementation does not esitamate parameters and uses a fixed value submitted by user as a $\alpha$ and log probability of a word given a topic as $\beta_{iw_n}$. We asked to the author of this implementation about effects of parameter estimation, he answered that it would not affect much for basic working of LDA. If we increase a topic smoothing parameter, it increases the effect of words that occur infrequently.

#### 4.1.2 Defect 2: Final output

The final output of LDA implementation is only the probability of a word given a topic, $p(w|k)$. We need to generates gamma file including document $\times$ topic matrix, which is variational posterior Dirhcilet distrubition. This result is necessary to infer the topic distribution for unseen documents using learned topic model.

#### 4.1.3 Convergence

The original algorithm proposed EM method to iterate until the lower bound of log likelihood value converges. E step is inference, and M step is maximizing lower bound of the log likelihood regarding the model parameters $\alpha, \beta$ using variational parameters obtained from E step. Mahout implementation does not execute M step and only repeats E step using previous calculated log probability of a word given a topic.

### 4.2 Yahoo! LDA

Due to above major drabacks of Mahout LDA implementation, we decided to use Yahoo! LDA implementation[4]. Yahoo! LDA implementation is not based on Hadoop MapReduce because MapReduce is a quite restrictive programming model and its disk IO between every job is not suitable for iterative machine learning representation. Yahoo! implemented distributed LDA computation using message passing way and it uses Hadoop Streamining and HDFS (Hadoop Distributed File System) as job communication and intermediate shared storage. We tried Yahoo! LDA for our meme collection but we failed to successfully run with the distributed environment on Longhorn cluster. Currently, Yahoo! LDA is operating with bash shell script for distributing work and synchronization and we reached to the conclusion that it is not fully packeged for distributed environment. The followings are problems we found during the experiment.

- Permission denited for excutables in jar package
- Synchronization problem of global/lda.dict.dump
- Ice::ConnectionLostException problem

We could resolve first two problems by fixing some lines of script files but we could not resolve the last problem. According to the late response of Yahoo! LDA author, lost connection problem is caused by Hadoop. Yahoo! LDA supports checkpointing and restart mechanism but when we restarted it, it started from the first iteration of LDA. The main difficulty was the large scale of meme collection and many iterations were needed for gaining meaningful topic collection. One iteration took average 7 minutes with a single machine and took average 1.5 minutes with 5 machines. Default number of iterations of Yahoo! LDA implementation is 1,000.

[4]https://github.com/shravanmn/Yahoo_LDA

### 4.3 Mahout Canopy Clustering

If we can represent our data as a vector, the most common and efficient clustering algorithm is $K$-means clustering. The main drawbacks of $k$-means clustering is that we have to specify the number of cluster $k$, a priori information of data distribution and $k$-means clustering is very sensitive to initial seed clusters. Thus, there are many suggestions that we can estimate suitable number of clusters and initial seed clusters using canopy clustering [4] because canopy clustering is generating clusters totally based on distance measure. Because Apache Mahout contains canopy clustering implementation, we implemented Jacadi distance measure to Apache Mahout interface and executed Mahout canopy clustering with some parameters. Canopy clustering receives 2 distance parameters $t_1, t_2$ such that $t_1 > t_2$. It starts with empty canopy list. If the distance between the vector and the canopy in the canopy list is less than $t_1$, the vector is added to the canopy. The distance from the canopy is the distance from the centroid of vectors in the canopy. If there is no canopy whose distance is less than $t_2$, which means the point is not strongly bounded for canopies, the new canopy is created with the point. This is repeated for every point. We have more than 4 million memes and they are highly sparse. Without any tuning of canopy clustering algorithm, even though we specify very generous parameters for $t_1, t_2$, we would have more than million canopies it caused out of memory error while running on Hadoop MapReduce. Also, since its time complexity is near $O(N^2)$ with many canopies, we could not use the original canopy clustering implementation. After these trials and errors, we decided to develop Indexed MapReduce Canopy.

### 4.4 Indexed MapReduce Canopy

Since the form of memes evolve over time across different media [3], similar memes need to be clustered together. For example, “lipstick look good on a pig,” “lipstick on the pig,” and ”lipstick on this pig” need to be seen in the same cluster. A canopy clustering algorithm [4] is used for this meme clustering. However, since we are clustering more than 4.6 million memes into many clusters of very fine granularity, different strategies should be adopted. Since Canopy is basically a single pass approach, the distance between every meme and every cluster needs to be calculated in $O(n^2)$ computational complexity. The meme clustering task is very different from an ordinary document clustering task as the number of documents is extremely high while the length of each document is usually fewer than a dozen of words. We devised the following optimizations and modifications to efficiently cluster 4.6 million memes:

- Binary vector: A vector is simply a set of words in a meme or a centroid. The set operation is O(1) while list operation is O(n). Also word order is ignored by this method.
- Centroid indexing: The centroid is indexed by the

words it includes so that when a meme is compared to only with the cluster centroids that include common words.

- New similarity coefficient: Jaccard similarity coefficient

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

is inappropriate for meme clustering. We compare a meme with a cluster centroid, and when a cluster centroid is long, short memes cannot have high coefficient scores. Therefore, instead of dividing by the union, we divide the number of common words by the average length of the two vectors:

$$memeSim(A,B) = \frac{|A \cap B|}{(|A| + |B|)/2}$$

The similarity score should be higher than 0.6 to be clustered into a given centroid.

- Stop word removal: The memes should be compared by meaningful words instead of stop words. Otherwise, "You can put lipstick on a pig" and "You can put a book on the table" would be clustered together. Using an extended stop word list that includes month names and weekday names, we excluded stop words from the similarity computation.

- Distributed computation: To cluster 4.6 millions of memes efficiently, mapreduce-based distributed Canopy approach, explained in Apache Mahout Canopy Clustering[5], is used. The mapper takes in a partial list of the memes. Using canopy algorithm, it generates cluster centroids. The reducer does the same clustering process with the aggregated centroids. The pseudo code of the MapReduce process is illustrated as follows:

**Meme-clustering:**
*Mapper:*
$CIndex \leftarrow newHashMap$ # Centroid Index
$C \leftarrow newList$ # Cluster Centroids
for $meme$ in $memes$:
  # Initialize $C$ and $CIndex$
  # 0 is the index of the first cluster
  if $C$ is empty:
    # set() makes a set of words in a meme
    $C$.append( set(meme) )
    for $word$ in $meme$:
      $CIndex[word]$.add(0)

  # Get candidate centroids from $CIndex$
  $Candidates \leftarrow newSet()$
  for $word$ in $meme$:
    for $c$ in $CIndex[word]$:
      $Candidates$.add($c$)

  $flag \leftarrow False$
  for $c$ in $Candidates$:
    if memeSim($meme$, $C[c]$)>0.6:
      $C[c]$=$C[c]$ + set($meme$)
      $flag \leftarrow True$
      for $word$ in $meme$:
        $CIndex[word]$.add($c$)
      break
  if $flag == False$:
    $C$.append(set($meme$))
    $c$=len($C$)-1
    for $word$ in $meme$:
      $CIndex[word]$.add($c$)
emit($C$)

*Reducer:*
The reducer is indentical to the mapper except that the input is the centroids($C$) instead of the memes($memes$).

[5]https://cwiki.apache.org/MAHOUT/canopy-clustering.html

# 5. MEME RANKING

Now that we have successfully clustered the memes, we still have many things to do in order to find valuable and meaningful information from the meme clusters. Essentially, we can classify every meme cluster as informative or not because there are many meaningless sequence of words such as "2 1956 jamboree tues jan 3 1956 jamboree wed,oct 5th wed oct 4 tues". If we extract some statistical features from meme clusters and build the training data indicating informativeness, we can predict informativeness of each meme cluster. Finally, if we build the model of classifier as a regression problem of probability of informativeness given a meme cluster, we can bring order into a set of meme clusters with the probability values. Currently, we are expecting the following features as discriminant to decide a meme cluster's informativeness.

- the number of memes in the cluster
- average meme length
- average number of documents in the cluster
- average number of sources in the cluster
- meme cluster cohesiveness
- time span information of blog documents containing memes in the cluster
  - the number of days blog documents were written
  - the number of documents per day

Meme cluster cohesiveness can be defined as

$$\frac{the\ number\ of\ unique\ words}{the\ number\ of\ words}$$

which means how similar memes in the cluster are each other.

## 5.1 Building Training Set at Amazon Mechanical Turk

Ranking the meme clusters based upon their meaningfulness or informativeness as described above is a difficult and largely subjective task. Consequently, using machine learning for this task is highly problematic. Instead, we are implementing an Amazon Mechanical Turk that will ask users to answer a set of basic yes/no questions about sample

meme clusters to determine their meaningfulness. Users will be asked questions such as "Is this a meaningful chunk of information?" and "Is this a common expression that could potentially appear elsewhere?" for approximately 1500 meme clusters. The composite score of each meme cluster will determine if it is meaningful or not. Using these labels we will build a training set of memes that can serve as input to the Support Vector Machine for automatic meme ranking.

## 5.2 Automatic Meme Ranking using Support Vector Machine

In training data, each meme cluster contains several scores by different evaluators. We remove the highest and lowest scores for each meme cluster to remove evaluator's biases. Then we calculate the mean score and normalize to 0-1 scale. This score will be the target value of ranking.

### *5.2.1 Classification using Support Vector Machine*

As a result of mechanical turnk, we get 1090 training instances. The first method for meme ranking is classification using Support Vector Machine. We classify meme clusters in the training set as positive or negative with the score threshold and then we calculate the probability that meme cluster in the test set is classfied as the positive set. If we sort meme clusters in the test set by the probability value, we can find significant meme clusters. We utilize the LIBSVM package[6] for the experiment. First of all, we have to find the threshold value to classify training instance as positive or negative. If the score is less than the treshold value, its class is negative otherwise positive. When we naively choose the threshold value as 0.5, our training data contains only 10% negative data and the trained model with this data predicts only 2 negative examples for the training data. So, we find the best threshold value which can balance between ratio of positive/negative data and prediction accuracy. Table 1 is showing the result for various threshold values. We use the tool included LIBSVM package to find the best parameters and feature scale.

As we can see from Table 1, threshold value 0.75 looks the best and other threshold value cannot learn to classify negative examples or positive examples at all. Armed with 0.75 as our threshold value, we calculated the probability of positive classification for each meme cluster and sort. Table 2 is showing the top 20 memes.

From the result, we can catch several keywords but it looks not so informative. When we search "lipstick on the pig" from the result, probability values of meme clusters containinig "lipstick on the pig" were all less than 0.5, which means these were all non-informative memes. We can analyze causes as the following:

- Errorneous feature engineering
- Reliability of Amazon Mechanical Turk training set
- Wrong approach of ranking

Because evaluating feature engineering and training set is not easy, we try to rank meme clusters using regression methid.

### *5.2.2 Ranking with Support Vector Regression*

Regression is estimating the target value based on feature values. Instead of estimatimg probability of meme cluster to be informative, if we calculate scores using Support Vector Regression (SVR), we can get informative meme clusters. We used $\epsilon$-SVR

# 6. EVALUATION

Based on the labels collected with the Amazon Mechanical Turk, the performance on SVM-based ranking will be evaluated.

# 7. CONCLUSION

# 8. LOGISTICS

## 8.1 Division of Work

- Hohyon Ryu: Data Preprocessing, Meme Extraction, Meme Clustering
- Jae Hyeon Bae: Theoretical Background, SVM-based Meme Ranking
- Nicholas Woodward: Theoretical Background, Crowdsourcing the meme cluster evaluation, Mock-up Webpage Build

## 8.2 Timeline

- Oct/6: Preprocessing - Done
- Oct/13: Meme Extraction - Done
- Oct/20 & 27: Meme Clustering - Done
- Nov/3 & 10: Crowdsourcing
- Nov/10: SVM-based Meme Cluster Ranking
- Nov/17: Finalizing Paper
- Dec/1: Presentation

# 9. REFERENCES


[1] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[2] O. Kolak and B. N. Schilit. Generating links by mining quotations. *Proceedings of the nineteenth ACM conference on Hypertext and hypermedia - HT '08*, page 117, 2008.

[3] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 497, New York, New York, USA, 2009. ACM Press.

[4] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, pages 169–178, 2000.


[6] http://www.csie.ntu.edu.tw/ cjlin/libsvm/

| threshold | No. of negative instances in training data | No. of negative instances predicted with the trained model | Cross-Validation accuracy(%) | training classification accuracy(%) |
|---|---|---|---|---|
| 0.5 | 108 | 2 | 90.27 | 90.27(984/1090) |
| 0.6 | 321 | 4 | 70.83 | 70.91(773/1090) |
| 0.7 | 427 | 38 | 61.38 | 62.11(677/1090) |
| 0.75 | 587 | 577 | 56.24 | 72.84(794/1090) |
| 0.8 | 639 | 1090 | 58.62 | 58.72(639/1090) |

**Table 1: Classification experiment using SVM**

| |
|---|
| also offering a 500,are you offering senate candidate 2,offering 10,offering 1,offering 20,offering 30,offering 3,offering 4,offering 5,offering 6 |
| am not 's design,decree of god,design to do's,god's decree,god's decree that those,god's design,it's design |
| 0 scientists,2004 scientists used,2008 scientists,400 scientists,january 2009 scientists have managed,the scientists used |
| 10 lessons,2008 lessons,2009 lessons,2 lessons,3 lessons,clarinet lessons 28,forgotten the lessons of 9 |
| my own room happy memorable 2007,room 101,room 1,room 205,room 222,room 2,room 401,room 6,room 8 |
| he's already gotten,he's gotten,it's gotten,she's gotten,that's gotten a lot of headline,what's gotten our shownotes,shownotes are complete,shownotes |
| 18 ct,1 slice,28 ct,32 ct,40 ct,6175 ne 195 ct,64 slice ct |
| it's sold,my soul's,sold at sotheby's,sold it's soul,s to be sold,the soul's,your soul's |
| energize a,energize,energize our,energize the,energize your,this would energize his supporters |
| a sultry,her sultry,sultry,the sultry,you are vixensexy and sultry |
| an exhortation,been my exhortation continually,exhortation,the exhortation |
| been snacking,be snacking,not too bad for snacking,snacking,while snacking |
| fear of rectum or,her rectum,his rectum,my rectum,the rectum,your rectum |
| cb,cb and so forth,cb,the cb |
| a centrist,conservative and centrist,more centrist,the centrist |
| it's required,required by today's,response to today's,that's required,what's required |
| a disgraceful,are disgraceful to ourselves,disgraceful,his disgraceful,offensive and disgraceful,the disgraceful |
| entitles him,entitles the holder,entitles them,entitles you |
| intimidate and terrorize,terrorize the,terrorize them,to terrorize |

**Table 2: Top 20 memes by classificatioin probability**