Scalable personalized trending videos for Youtube

Dhrumil Shah (201301034) DAIICT
Sumeet Varma (201301402) DAIICT

***Abstract*—Youtube has a page called trending videos which shows "The most random, yet interesting videos recently published on the web!". However, it is not personalized for each user. In this paper, we provide a scalable algorithm to find and show trending videos to each user based on his/her interests and what videos he/she has watched in the past. Our algorithm for recommending the videos works only in one pass to all videos user has watched and it is scalable. We observe that the jaccard similarity of descriptions of two videos as similarity of both videos increases. We use this fact to cluster videos, and based on this clusters and user's past browsing history, we recommended trending videos to user.**

I. Introduction

Youtube has a feature called trending videos but the challenge is to find right content for yourself. The right content is defined as something that answers your current information need or something that you would love to watch and listen. However in impersonalized trending videos, users may not even know what to look for. So the aim of our project is to provide personalized trending videos for youtube.
The task can be divided in two steps:

```
1) Clustering recent published videos
   based on their similarity.
2) Finding personalized trending score For
   each video using:
   a) Clusters formed in above step.
   b) Users past browsing history
   c) Quick rise in views of video
```

II. Similarity of videos for clustering

We define cluster to be a subset of videos, such that all the videos in a same cluster are highly similar. Also, Videos coming from different cluster will have less similarity. To define clusters, we need to define similarity between videos. We can define similarity of two videos based on the words in title and description. To do this, we have taken only adjectives and nouns in title and description as tags for the video. So each video can be represented as a list of tags. We used jaccard similarity to measure similarity between two videos. Jaccard similarity of two videos now can be defined as:

$$JaccardSimilarity(V_i, V_j) = \frac{V_i \cap V_j}{V_i \cup V_j}$$

Where, V_i is set of tags of i^{th} video
The reasons behind using jaccard similarity and not any other measure are as follows. Cosine similarity of any two videos will be nearly equal to zero due to large number of distinct tags present. We cannot measure euclidean distance also because every video in this huge dimensional space is more or less at equal distance from origin. We observe that similarity between two videos increases with increase in jaccard similarity of tags. It is manifested in following graph.

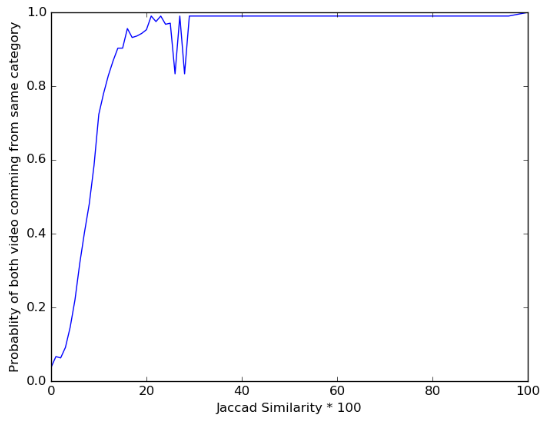

Figure 1: Jaccard Similarity vs Probability that videos coming from same category

III. Clustering videos based on Jaccard similarity

To cluster videos, we used min-hash clustering algorithm. Min-Hashing is a probabilistic clustering method that assigns a pair of videos to the same cluster with probability proportional to Jaccard similarity between two videos. The key idea is to hash the data points using several hash functions so as to ensure that, for each function, the probability of collision is much higher for objects which are close to each other than for those which are far apart. Then, we can determine near neighbors by hashing the query point and retrieving elements stored in buckets containing that point i.e. elements having same hash. These buckets can be thought of as clusters.

Let h be a hash function that maps the tags of V_i and V_j to distinct integers where V_i is set of tags of i^{th} video, and for any set of tags S, define $h_{min}(S)$ to be the minimum hash value of tags in S with respect to hash function h . Now, if we apply h_{min} to both V_i and V_j, we will get the same value exactly when the element of $V_i \cup V_j$ with minimum hash value lies in $V_i \cap V_j$. The probability of this being true is the ratio below, and therefore it is same as $JaccardSimilarity(V_i, V_j)$

$$Prob[h_{min}(V_i) = h_{min}(V_j)] = \frac{V_i \cap V_j}{V_i \cup V_j}$$

$$= JaccardSimilarity(V_i, V_j)$$

Clearly, defining clusters based on only one hash function, will not give accurate results. So we will use set of independent hash functions to cluster videos. To analyze the effect of set of independent hash functions, we define Families of hash function and their sensitivity.

IV. Families of hash function and their sensitivity

Let say we have a space S of points with a distance measure d.Family H of hash functions is said to be $(d1, d2, p1, p2) - sensitive$ if for any x and y in S, if $d(x, y) < d1$, then probability over all h in H, that $h(x) = h(y)$ is at least p1 and if $d(x, y) > d2$, then prob. over all h in H, that $h(x) = h(y)$ is at most p2. Let S = set of tags and d = Jaccard distance. Here Jaccard distance is 1 - Jaccard similarity, H is formed from the minhash functions for all permutations.

$$Prob[h(x) = h(y)] = Jaccard_Similarity(x, y)$$

$$= 1 - Jaccard_Distance(x, y)$$

Therefor using jaccard similarity, min-hashing gives us a $(d1, d2, (1 - d1), (1 - d2))$ sensitive family for any $d1 < d2$

AND of Hash Functions: Lets say we are given a family of Hash functions H. We can construct a family of Hash functions H' using r hash functions from H as below:
Let $h = [h_1, h_2, ..., h_r]$ in H', where all h_i's are independent chose hash function from H. And $h(x) = h(y)$ if and only if $\forall_i h_i(x) = h_i(y)$. Now, probability of collision of hash function can be given as:

$$Prob[h(x) = h(y)] = (\frac{x \cap y}{x \cup y})^r$$

Therefor if sensitivity of H is (d1, d2, p1, p2), then sensitivity of H' is $(d1, d2, p1^r, p2^r)$.
AND of hash functions makes hash collision probabilities shrink, but by choosing r correctly, we can make the lower probability approach 0 while not much affecting the higher.

OR of Hash Functions: Lets say we are given a family of Hash functions H. We can construct a family of Hash functions H' using r hash functions from H as below:
Let $h = [h_1, h_2, ..., h_r]$ in H', where all h_i's are independent is a hash function from H. And $h(x) = h(y)$ if and only if $\exists_i h_i(x) = h_i(y)$. Now, probability of collision of hash function can be given as:

$$Prob[h(x) = h(y)] = 1 - (1 - (\frac{x \cap y}{x \cup y}))^r$$

Therefor if sensitivity of H is (d1, d2, p1, p2), then sensitivity of H' is $(d1, d2, 1 - (1 - p1)^r, 1 - (1 - p2)^r)$.
OR of hash functions makes all probabilities grow, but by choosing r correctly, we can make the upper probability approach 1 while not affecting lower probabilities.

AND-OR composition: Lets say we are given a family of Hash functions H. We can construct hash function H' using R*B hash functions of H.

Let $h = [h_{11}, h_{12}, ..., h_{1r}, h_{21}, ..., h_{2r}, ..., h_{b1}, h_{b2}, ...h_{br}]$ in H', where all h_{ij}'s are independent is a hash function from H. And $h(x) = h(y)$ if and only if $\exists_i \forall_j h_{ij}(x) = h_{ij}(y)$. Now, probability of collision of hash function can be given as:

$$Prob[h(x) = h(y)] = 1 - (1 - (\frac{x \cap y}{x \cup y})^r)^b$$

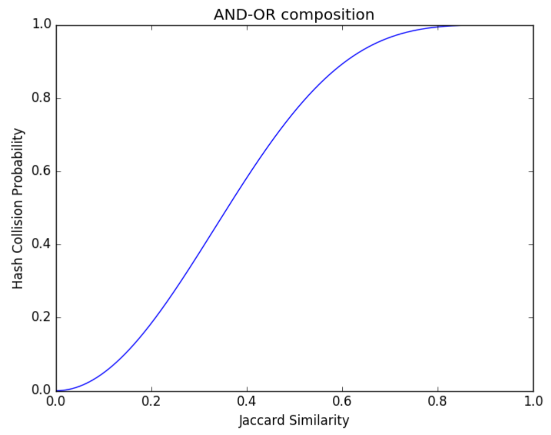

Figure 2: Jaccard Similarity vs Hash Collesion Probablity

We can see that, function is monotonically increasing. For R = 1 and B = 1, it will produce a line with slope 1. When we increase value of R, probabilities in lower half will decrease. When we increase value of B, probabilities in upper half increases. Using appropriate values of R and B, such that hash collision probability is increased above the threshold value of jaccard similarity, and decreases below threshold value.Since the graph is monotonically increasing, threshold value can be determined using this formula $1 - (1 - t^r)^b = t$. In our case, threshold value is 0.39

V. Preparing Data

We used youtube data api v3 provided by google to take the video description and title. we have used keyword based search to find video ids. we have chosen 5 topics cricket, news, movie trailer, technology, comedy. Here please note that due to limitation of youtube api we are not able to take more then 50 videos for each topic. After that we have taken description of video and split it into words using StringTokenzer of java and convert it in to tags using natural language processing library of stanford core-nlp porter stemming for stemming a string and created a hashset of words for each video to calculate Jacard similarity between videos. Here please note that numbers are also excluded from data set.

VI. Implementation Detail of Minhash Algorithm with AND-OR composition

The crucial part is to generate k independent hash functions to generate k independent hash functions an for that we used the following algorithm.

Pseudo Code to generate k independent hash functions

```
1) Generate k random numbers and store
    them in seed[]
2) For i in range(1,k) :
   h[i] = hash(concatenation(seed[i], tag,
        seed[i]));
```

where concatenation(x,y,z) = xyz where x, y, z are strings. and hash(a) = hash of string a
To efficiently implement AND composition, we appended r hash values together and for OR composition we hashed one video b times. To implement AND-OR composition we hashed one video b times every time we used AND composition of r hash values to hash it.

Minhash Algorithm with numAnd AND and numOr OR composition pseudo code :

```
for (video in videos)
   for (i in range(0, numOr))
      h(i) = ""
      for (j in range(0, numAnd)
         minHash = 0
         for (tag in video.tags)
            minHash = min (minHash,
                hash(seed[i][j]+tag+seed[i][j])
      h(i) = h(i).append(minHash)
   emit(h(i), video)
```

As we can see, the process is independent for each video and thus we implemented it parallely in Scala which runs 1 thread for each video. After running the above algorithm, we have a set of clusters where each cluster is a set of videos. Note: A video will be present in at max numOr clusters and at least 1 cluster.. We pruned the clusters having less than 5 videos using Another minhash with reduced no of AND composition. This was done to avoid clusters having very small size which would add little to our recommendation engine.

VII. Recommendation strategy using clusters

To recommend a video to a user let's first define cluster score for the user based on his watch history. User watched n videos and we have m clusters. Cluster score for user as follows :

$$clusterScore[i] = \frac{\sum_{j=1}^{n} I_i(j)}{n} + \epsilon$$

where $I_i(j) = \begin{cases} 1, \text{ if } j^{th} \text{ video is in cluster i} \\ 0, \text{ otherwise} \end{cases}$

Let say there are N recently published videos then video score for those videos ca be defined as follows

$$videoScore[i] = \sum_{j=1}^{m} clusterScore[j] * I_j(i)$$

Video score gives the user's interest in the video.
Velocity of the video can be defined as follows

$$velocity[i] = \frac{viewCount_{12}[i]}{viewCount_{24}[i]}$$

Where $viewCount_X[i]$ = view count of i^{th} video in last X hours. Here velocity is proportional to last12 hour view count and inverse proportional to last 24 view counts. velocity reflects the polpularity of the video in recent.

$$recommendationScore[i] = videoScore[i] * velocity[i]$$

Recommendation score is to proportional to both video score and velocity, as video score gives the amount of user's interest in the video and velocity gives recent trend for that video.

We will sort recent published videos based on recommendation score, and recommend top X videos to user. In our case, we used X = 5.

VIII. Evaluation

We have used 3 matrices to evaluate clustering algorithm.

1) **Precision**:
Precision is defined as:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

In our algorithm of clustering, We got precision of 20.7%

2) **Recall**:
Recall is defined as:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

In our algorithm of clustering, We got recall of 90.31%

3) **Accuracy**:
Accuracy is defined as:

$$Accuracy = \frac{TruePositive + TrueNegative}{TruePositive + TrueNegative + FalsePositive + FalseNegative}$$

In our algorithm of clustering, We got Accuracy of 20%

4) **F1-Score**:
F1-score is defined as:

$$F1score = \frac{2 * Precision * Recall}{Precision + Recall}$$

In our algorithm of clustering, We got f1-score of 32.92%

True Positive = Number of pairs of video which are in

same category and shares at least one cluster
True Negative = Number of pairs of video which are in different category and do not share any cluster
False Positive = Number of pairs of video which are in different category and shares at least one cluster
False Negative = Number of pairs of video which are in same category and do not share any cluster

5) **Purity Matrix**
Purity matrix is defined as follow:

$$Score_{Cluster_i} = \frac{\sum_{i=1}^{Size_{Cluster_i}} \sum_{j=i+1}^{Size_{Cluster_i}} JaccardSimilarity(i,j)}{\binom{Size_{Cluster_i}}{2}}$$

$$PurityMatrix = \frac{\sum_{i=1}^{m} Score_{Cluster_i}}{m}$$

In our algorithm of clustering, We got Purity Matrix value of 32.80%. Here note that threshold was 0.39 which is very close to 0.3280 .

IX.CONCLUSION

Here we have presented the algorithms behind scalable real time recommendation engine and gave the results of its evaluation which were positive. We presented our approach for clustering over dynamic data-sets using Min-Hash which is adapted to scale arbitrarily due to inherently parallel nature of Min-hash functions. Note that we did not have to trade between scalability and quality as have achieved both because of AND OR composition. Thus, using our algorithm and data mining strategies we have successfully built a scalable recommendation engine to suggest personalized trending videos to each user.