Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph

Shumeet Baluja Rohan Seth D. Sivakumar
Yushi Jing Jay Yagnik Shankar Kumar Deepak Ravichandran Mohamed Aly *
Google, Inc.
Mountain View, CA, USA
{shumeet, rohan, siva, jing, jyagnik, shankarkumar, deepakr}@google.com,
midohussein@gmail.com

ABSTRACT

The rapid growth of the number of videos in YouTube provides enormous potential for users to find content of interest to them. Unfortunately, given the difficulty of searching videos, the size of the video repository also makes the discovery of new content a daunting task. In this paper, we present a novel method based upon the analysis of the entire user–video graph to provide personalized video suggestions for users. The resulting algorithm, termed *Adsorption*, provides a simple method to efficiently propagate preference information through a variety of graphs. We extensively test the results of the recommendations on a three month snapshot of live data from YouTube.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*

General Terms

Algorithms

Keywords

Recommendation systems, label propagation, collaborative filtering, random walks, video search

1. INTRODUCTION

Since the launch of YouTube in 2005, it has become a popular destination site for users to find videos as well as share their own videos. It is estimated that there are over 45,000,000 videos in the repository [14], and that the collection is growing at an astounding rate of seven hours of video being uploaded every minute [6]. This enormous repository of video information has the potential to contain videos of interest for many users. The downside to the quantity of videos is that exploration and discovery of new, interesting, videos becomes a daunting task. Standard approaches for recommendations that have been used in text domains, such as newsgroup, news story or web pages, are not easily applied in this domain. The primary difficulty is that although some labels can be reliably inferred through computer-vision based-techniques, there does not currently exist any satisfactory mechanism to label videos with the majority of their content [12]. To exacerbate the difficulty, the tags that exist on YouTube videos are generally quite small; they only capture a small sample of the content.

The task of providing valuable suggestions of non-text content has been explored in a variety of contexts. The closest related studies have come from the Netflix challenge, in which a system must recommend DVDs to subscribers based on their previous ratings [11]. The Netflix domain has under 100,000 DVDs, and Netflix users provide a large number of explicitly given ratings on the videos. In contrast, for our task, the number of videos is significantly larger, and the ratings are sparse. Our approach to addressing this task is to utilize the large number of users and video views that YouTube has amassed. By studying the viewing patterns and video discoveries of YouTube users in aggregate, we can create an effective video suggestion system that does not rely on the analysis of the underlying videos. The goal is to create a personalized page of video recommendations that not only shows the latest and most popular videos, but also provides users with recommendations tailored to their viewing habits.

In the remainder of this section, we introduce one of the principal data-sources used in this study — co-view statistics. In Section 2, we present the *adsorption* algorithm, and describe several related studies that have been proposed in the machine learning literature. The adsorption algorithm is a very general framework for classification and learning when we have some labeled objects and a rich graph structure that underlies the universe of labeled and unlabeled objects. The algorithm is also robust in the mathematical sense that it has three different but equivalent interpretations, and has appeared in various guises in applied mathematics and machine learning. In Section 3, we present the data used for this study. The evaluation of the proposed systems is itself a challenging task; each system must be thoroughly vetted before being deployed into live production. Section 3.1 explores the use of historic data to provide the necessary insights into algorithm performance, before live-service launches. The results of the experiments are given in Section 4. Section 5 closes the paper with conclusions and directions for future research.

*Work done as an intern at Google while a student at the University of Pittsburgh.

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

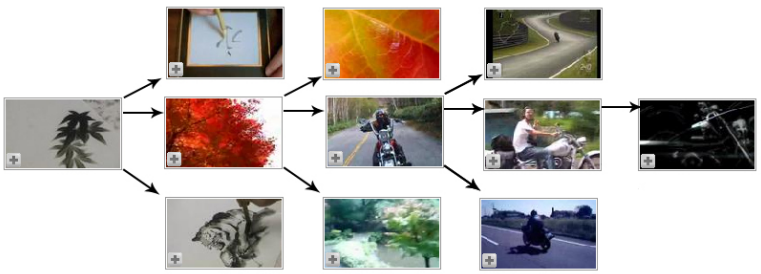

Figure 1: Video-Video Co-View Graph. Each video is a vertex in the graph that is linked to other videos often co-viewed. Often, only links with some minimum number of views are instantiated.

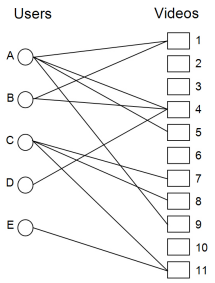

Figure 2: User-Video Graph. An alternate way of representing the co-view information is implicitly through the user-video bipartite graph. The number of co-views is computed by examining the number of paths of length 2 that exist between any two videos.

1.1 The Video Co-View Graph

YouTube has a large number of users who view multiple videos. One of the fundamental set of statistics to compute with this data are the video co-view numbers. In the simplest form, the co-view data gives, for any pair of videos, the number of people who viewed both videos. After this statistic is computed for all sets of videos, there are numerous ways to encode this into a graph. Two that are used in this study are shown here. In Figure 1, we show the "video-video co-view graph," which has a connection between videos that are most commonly watched by the same users. In Figure 2, we show the user-view graph, from which co-views can be inferred [1].

Co-view information provides a simple basis for video recommendations. A straightforward system, often used as the basis of *item-based* collaborative filtering systems [5, 1], can be built as follows. Imagine that User U watches two videos, J and K. From our co-view statistics, we know that many other users who saw video J also saw videos L, M, N. Similarly, for video K, we know that many other users saw videos N, O, P, Q. Therefore, the videos we may recommend to U, based on his watches of J and K can simply be the union of the two co-view sets: L, M, N, O, P, Q. For ranking the recommendations, we may look at the number of views of a video (this will recommend popular videos), number of co-views for each video (this will recommend popular videos, given what users have seen), or take into account the number of times each video was recommended for U (notice that Video N was recommended twice), or combinations of any of these heuristics.

It should be noted that there are many versions of co-view statistics and graphs that can be used in place of the ones described above. A stronger notion of similarity can be captured by restricting the co-views to those that have occurred within the same web session. This alternate view may be useful if it is believed that a user will be more likely to view videos on the same topic within a short period of time (for example, watching multiple instructional videos on how to draw in the sumi-style, etc). Other potential co-view variations include encoding ordering information through the use of directed edges. The drawback of using any of the many variations is that they often result in smaller amounts of data per video/user. Given the large number of videos for which we need to compute co-view statistics, we attempt to use the broadest definition of co-view whenever possible.

[1]It is interesting to examine the large changes in subject matter that occur by traversing, even a short distance, along the edges in either of the graphs. For example, in Figure 1, Left-Most: Videos about Sumi Drawing are related to other Sumi-Drawing demonstrations and a video of a common subject matter from the videos: leaves, flowers, etc. From the red-leaf video, other videos about exploring nature and road-trips are found. Following the road-trip co-views, we find videos about road-trips on motorcycles. Following a co-view from a motorcycle road-trip, we are led to a video purely about motorcycles (right-most).

2. THE ADSORPTION ALGORITHM

The genesis of the family of algorithms that we will collectively call *adsorption* is the following question: assuming we wish to *classify* a node in a graph in terms of (class) labels present on some of the other nodes, what is a principled way to do it? Perhaps the easiest answer to this question is to impose a metric on the underlying graph and classify the label by adopting the labels present on its nearest neighbor. There is a variety of metrics one could choose from (e.g., shortest distance, commute time or electrical resistance, etc.), but most of these are expensive to compute, especially for large graphs. Furthermore, conceptually simple ones like shortest distance have certain undesirable properties in the context of recommendation systems. Based on the user–video view graph, if we recommend to user u a video v that she has not watched and that is closest in terms of graph distance, we may end up recommending a node that can only be reached via high-degree nodes (for example, if users u_1 and u_2 both watched a popular video, even though they have no interests in common, we will end up recommending videos watched by u_2 to user u_1, since there will be length-3 paths from u_1 to those videos, and this is the shortest possible distance to any of u_1's unwatched videos. More sophisticated methods like commute distance avoid this problem, but require expensive computations; furthermore, these methods do not lead to algorithms that admit incremental updates, another factor important for large recommendation systems.

The idea of recommending videos *commonly* co-viewed with a user's watched videos nevertheless can be meaningfully abstracted as finding labeled nodes that have *multiple short paths* from the user node. In doing so, one must be careful to not veer too far off from the node, for if the only paths connecting a user to a video are long, there is a considerable chance of drift of interest and relevance.

Thus our desiderata for our recommendation system include the ideas that a video v is relevant to a user u if:
(1) u and v have a short path between them;
(2) u and v have several paths between them;
(3) u and v have paths that avoid high-degree nodes.

2.1 Adsorption via Averaging

The first view of our algorithm that we present has the following idea. In a general graph where some nodes have labels, the nodes that carry some labels, forward the labels to their neighbors, who, in turn, forward them to their neighbors, and so on, and all nodes collect the labels they receive. Thus each node has two roles, forwarding labels and collecting labels. In the "full-information" model, let us imagine that each node keeps track of the history of all labels it receives — how often it received it, in which rounds it received it, etc. This would enable each node to make an informed choice of which labels it wishes to retain. The crucial detail in the algorithm is the choice of how to retain a synopsis that will both preserve the essential parts of this information as well as guarantee a stable (or convergent) set of label assignments.

A formal description follows. We are given a graph $G = (V, E, w)$, where V denotes the set of vertices (nodes), E denotes the set of edges, and $w : E \to \mathbf{R}$ denotes a non-negative weight function on the edges. Let L denote a set of labels, and assume that each node v in a subset $V_L \subseteq V$ carries a probability distribution L_v on the label set L. We often refer to V_L as the set of labeled nodes. For the sake of exposition, we will introduce a pre-processing step, where for each vertex $v \in V_L$, we create a "shadow" vertex $\widetilde{v}$ with exactly one out-neighbor, namely v, connected via an edge $(\widetilde{v}, v)$ of weight 1; furthermore, for each $v \in V_L$, we will re-locate the label distribution L_v from v to $\widetilde{v}$, leaving v with no label distribution. Let $\widetilde{V}$ denote the set of shadow vertices, $\widetilde{V} = \{\widetilde{v} \mid v \in V_L\}$. From now on, we will assume that at the beginning of the algorithm, only vertices in $\widetilde{V}$ have non-vacuous label distributions.

Algorithm Adsorption:

Input: $G = (V, E, w)$, L, V_L.
repeat
  for each $v \in V \cup \widetilde{V}$ do:
    Let $L_v = \sum_u w(u, v) L_u$
  end-for
  Normalize L_v to have unit L_1 norm
until convergence
Output: Distributions $\{L_v \mid v \in V\}$

Comments.

(1) In the summation, u varies over all vertices in $V \cup \widetilde{V}$ that have a non-vacuous label distribution L_u.

(2) In the algorithm, we say that convergence has occurred if the label distribution of none of the nodes changes in a round. It can be shown that the algorithm converges to a unique set of label distributions. In practice, we will impose a small threshold so that if none of the distributions undergoes a change of magnitude greater than this threshold, we will say that the algorithm has converged.

(3) Upon convergence, each node $v \in V \cup \widetilde{V}$ carries a label distribution, provided there is a path from v to some node $u \in V_L$.

(4) The choice of unit weight on the edge $(\widetilde{v}, v)$ for each $v \in V_L$ is entirely arbitrary, and may be replaced by other interesting choices; this will turn out to be a very useful feature.

(5) The astute reader might have noticed that we do not *update* the label distribution in each round; rather, we *recompute* it entirely from scratch, based on the distributions delivered by the neighbors. It turns out that the algorithms are entirely equivalent, and the memoryless property of the algorithm as presented makes it easier to analyze mathematically.

(6) The Adsorption algorithm admits a very efficient iterative computation (similar to PageRank [3]), where, in each iteration, a label distribution is passed along every edge. This is also an operation that is efficient to implement in the MapReduce model/infrastructure for parallel programming [8].
End of comments.

2.1.1 Related work

Recalling that our goal was to maintain a synopsis of the labels that are reachable from a vertex, let us remark that the normalization step following the step of computing the weighted sum of the neighbors' label distribution is crucial to our algorithm. Labels that are received from multiple (or highly-weighted neighbors) will tend to have higher mass after this step, so this step renders the adsorption algorithm as a classifier in the traditional machine learning sense. Indeed, the algorithm, as presented is a modification of the label propagation algorithm of Zhu and Ghahrahmani [16, 15], one of the first papers in the machine learning literature to consider the problem of semi-supervised classifier design using graphical models. Zhu and Ghahrahmani also note that their algorithm is different from a random-walk model proposed by Szummer and Jaakkola [13]; as we shall see, there is a very different random walk algorithm that coincides exactly with the adsorption algorithm. The latter fact has also been noticed independently by Azran [2].

From a scientific standpoint, however, this family of "repeated averaging" algorithms have a long history in the mathematical literature of differential equations, specifically in the context of boundary value problems. An archetypal problem in these areas is to estimate the heat at various points of a laminar surface superimposed on a 2-d grid, given its temperature at the boundaries; the most common algorithm here is to repeatedly average the values from the grid neighbors until the temperatures converge. Indeed, the natural generalization of this is to replace a grid by an arbitrary graph, with the labeled nodes being the analogue of the boundary points. However, a graph, in general, is not a continuous structure and one has to deal with various anomalies that arise because of this.

2.2 Adsorption via Random Walks

The memoryless property of the adsorption algorithm that we alluded to earlier immediately leads to a closely related interpretation. Let us "unwind" the execution of the algorithm from the final round, tracing it backwards.

For a vertex $v \in V$, denote by N_v the probability distribution on the set of neighbors of v described by $N_v(u) = w(u, v)/(\sum_u w(u, v))$, that is, the probability of u is proportional to the weight on the edge (u, v). The label distribution of a vertex v is simply a convex combination of the label

distributions at its neighbors, that is, $L_v = \sum_u N_v(u) L_u$; therefore, if we pick an in-neighbor u of v at random according to N_v and sample a label according to the distribution L_u, then for each label $\ell \in L$, $L_v(\ell)$ is precisely equal to $\text{Exp}_u[L_u(\ell)]$, where the expectation arises from the process of picking a neighbor u according to N_v. Extending this to neighbors at distance 2, it is easy to see that for each label $\ell \in L$, $L_v(\ell) = \text{Exp}_w \text{Exp}_u[L_w(\ell)]$, where an in-neighbor u of v is chosen according to N_v and an in-neighbor w of u is chosen according to N_u. Expanding this out, we see that

$$L_v(\ell) = \sum_w \sum_u N_v(u) N_u(w) L_w(\ell).$$

Notice that $N_v(u)$ is the probability of reaching u from v in one step of a random walk starting from v and picking a neighbor according to N_v, and similarly, $N_u(w)$ is the probability of picking a neighbor w of u according to N_u. Notice also the crucial use of the Markov property (memorylessness) here, that is, conditioned on the random walk having reached u, the only information used in picking w is N_u, which depends only on u, and not on where we initiated the random walk from.

Finally, note that if the random walk ever reaches one of the shadow vertices $\widetilde{z}$, where $z \in V_L$, then there is no in-edge into z, so the random walks stops. Thus vertices in $\widetilde{V}$ are "absorbing states" of the Markov chain defined by the random walk. A simple induction now establishes that the adsorption algorithm is equivalent to the following variation, described in terms of random walks on the *reverse* of the graph G together with the edges from $\widetilde{V}$ to V.

In our exposition below, the algorithm takes a starting vertex v for the random walk, and outputs a label distribution L_v for it when it reaches an absorbing state. Thus the label distribution for each node is a random variable, whose expectation yields the final label distribution for that vertex. To obtain label distributions for all vertices, this procedure needs to be repeated several times for every vertex, and the average distributions calculated. Clearly, this yields a very inefficient algorithm; therefore, in practice, we exploit the equivalence of this algorithm to the averaging Adsorption algorithm in Section 2.2, which has a very efficient implementation, especially in parallel computing models.

Algorithm Adsorption-RW

Input: $G = (V, E, w)$, L, V_L, distinguished vertex v.
Let $\widetilde{G} = (V \cup \widetilde{V}, E \cup \{(v, \widetilde{v}) \mid v \in V_L\}, w)$.
Define $w(v, \widetilde{v}) = 1$ for all $v \in V_L$.
done := false
vertex := v
while (not done) do:
  vertex := pick-neighbor(v, E, w)
  if (neighbor $\in \widetilde{V}$)
    done := true
end-while
u := vertex
Output label according to L_u.

Here, pick-neighbor(v, E, w) returns a node u such that $(u, v) \in E$ (so that there is an edge from v to u in the reversed graph) with probability $w(u, v)/(\sum_u w(u, v))$.

It is instructive to compare algorithm Adsorption-RW with typical uses of stationary distributions of random walks on graphs, such as the PageRank algorithm [3]. In cases like PageRank, a fixed Markov random walk is considered, therefore the stationary probability distribution gives, for each node of the graph, the probability that the walk visits that node. In the absence of any absorbing node (and assuming the walk is ergodic), the initial choice of the node from which the random walk starts is completely irrelevant in determining the probability of reaching any particular node in the long run. Consequently, these methods do not allow us to measure the influence of nodes on each other. In our situation, by contrast, labeled nodes are absorbing states of the random walk; therefore, the starting point of the walk crucially determines the probability with which we will stop the walk at any of the absorbing states. This implies that we may use these probabilities as a measure of the influence of nodes on each other.

2.3 Adsorption via Linear Systems

A third view of the algorithm emerges by observing that the averaging algorithm automatically implies that at each node v, the final distribution L_v on the label set L is a convex combination of the L_u's, where $u \in V_L$ (transferred to its shadow $\widetilde{u}$).

Indeed, one may attempt to describe *every* node of a graph as a convex combination of the other nodes in terms of how similar they are (proximity, neighborhood overlap, etc.), regardless of whether there any labels, etc. This is a very general problem with potentially numerous applications (see, e.g., [7], for some). In fact, doing so would give an embedding of the vertices of the graph into L_1 (since each convex combination can be written as a vector of unit L_1 norm in n dimensions, where $n = |V|$); this is a topic of active research in computer science (see [9]). The embeddings resulting from adsorption are not intended to capture shortest distances (which is a primary concern in the literature), but have very closely related and useful properties. A useful side benefit of the embeddings produced via adsorption is that there is a very natural notion of "continuity" of the embedding: the image of any node in L_1 is a convex combination of its neighbors' images. To the best of our knowledge, such embeddings have not been studied in the literature, and might be very useful in certain applications.

It can be shown that by casting the adsorption algorithm as a system of linear equations, one can obtain precisely such an embedding. Furthermore, the system of linear equations has a unique solution if and only if the underlying graph is connected. This leads us to the third algorithm for the label propagation problem on graphs, described in this more general framework. Once we express each vertex as a convex combination of the other vertices, we may obtain a label distribution for any vertex by taking a suitable (possibly scaled) convex combination of the label distributions at the labeled vertices.

Algorithm Adsorption-LS

Input: $G = (V, E, w)$
Let $n := |V|$
Define the linear system of equations in n^2 variables X_{uv}, for $u, v \in V$:

$$\sum_v X_{uv} = 1 \qquad \forall u \in V;$$

$$\sum_{z:(z,u)\in E} w(z, u) X_{zv} = X_{uv} \qquad \forall u, v \in V.$$

The linear system viewpoint offers another natural algorithmic approach for label propagation. In addition, it offers very natural ideas for obtaining computationally efficient versions of the algorithm. For example, we might restrict $X_{uv} = 0$ if u and v do not have any path of length at most t, for some parameter t. This has the effect of sparsifying the linear system of equations, which helps solve it more efficiently. For the video recommendation algorithm, this also has a nice semantic interpretation: do not recommend a video to a user if no other user in a ball of radius of t from the user has watched it; this helps curb the propagation of videos that are popular in communities distant to that of a user, regardless of how popular they are.

Another benefit of the viewpoint in terms of linear system of equations is that incremental updates to the label distributions or addition or deletion of nodes can be easily accommodated by quickly updating the information for the relevant neighborhood of the graph.

We conclude this section with the main mathematical statement, asserting the equivalence of the three adsorption algorithms.

THEOREM 1. *Algorithms Adsorption, Adsorption-RW, and Adsorption-LS are equivalent.*

2.4 Injection and Dummy Probabilities

The three equivalent renditions of the algorithm lead to a number of interesting variations that one may employ. As already noted, the viewpoint of linear system of equations allows us to restrict which labels are allowed for a given node. We now point out other interesting variations one may obtain by taking advantage of the alternative interpretations.

Recall the notion of "shadow" nodes $\widetilde{v}$ that acts as a "label-bearer" for v. A judicious choice of edge weight along the edge $(\widetilde{v}, v)$ (equivalently, a choice of transition probability from v to $\widetilde{v}$ in the reversed graph) helps us control precisely how the random walk behaves; this has a very useful application. Consider the application of video recommendation — suppose we start a random walk at a user u, and traveling along edges, arrive at a video node v. Now the probability with which we enter the absorbing state $\widetilde{v}$ might be determined as a function of various features of v — its popularity, freshness, community interest (in terms of discussion, tagging, etc.), reputation of the user who uploaded it, etc. Thus, this parameter, which we call the *injection probability* is usually a crucial design choice in deploying a live recommendation system. Similarly, in other applications, we have found that insightful choices of this probability often plays a significant role in the quality of results.

The next parameter that is very useful in controlling the behavior of the algorithm is another novel exploitation of our equivalence theorem. Namely, instead of considering the standard random walk on an edge-weighted graph, one may consider a "hobbled walk," where at each step, with some probability, which we call the *abandonment probability* or the *dummy probability*, the algorithm abandons the random walk without producing any output label. This is extremely useful, for example, when the random walk visits a high-degree node. Consider, for instance, a random walk originating at a user node u in the context of video recommendations. Suppose the random walk, after a few steps, arrives at a node z in the graph. If z is a video node of high degree (extremely popular video), taking another step will almost surely lead the random walk to users who are quite likely very different from u (since popular videos tend to be watched by users with no identifiable notion of shared interest). Similarly, if z is a user node of high degree (voracious user), taking another step might lead to a video that user u does not care much about (considering that z likely has a very broad spectrum of interests). To seamlessly integrate this feature into the algorithm, we introduce the idea of a "dummy" label, and modify the label distribution at each node to include the "dummy" label with suitable probability (usually reflecting the degree of the node).

Our experiments (in this paper and in other applications) have confirmed that adding a dummy label (equivalently, abandoning the random walk selectively) is indeed a very useful feature. It has the interesting side effect of slowing down the random walk in a quantifiable way: the influence of a label ℓ on a node u falls off exponentially in the number of labeled nodes along paths from nodes that carry ℓ to u.

3. EXPERIMENTAL SETUP

The raw data for our experiments was collected from live user views of videos from `youtube.com` over 92-day period from July 1, 2007, through September 30, 2007. We picked a sample of approximately 5.4 million users from a specific geographic area (so that language, broad subjects of interest, etc., are likely to be similar). For each of the chosen users, we collected the list of all videos they watched beyond the 33% mark; this restriction was applied to approximately confirm that the user actually liked the video. This resulted in nearly 29 million total views of a set of approximately 4.2 million videos (some of which may be duplicates). All the data was extracted in suitably anonymized form, so we were dealing with a bipartite graph of users and videos (as in Figure 2).

We partitioned the data into two sets — a training set consisting of all watches from the first 46 days of the evaluation period, and a test set consisting of all watches from the remaining days in the evaluation period. The exact way in which we used these, and the metrics we employed will be outlined in Section 3.1. The training set was used to run all of the recommendation algorithms and to derive the recommendations based on those watches; the effectiveness of these recommendations is measured against the test set. We consider a recommendation of video v to user u successful if user u had not watched video v in the training period, but did watch video v in the test period. As is usually done in information retrieval applications, we will measure (variants of) precision and recall (respectively, whether the user watched what the algorithm recommended, and whether the algorithm recommended everything the user watched). Before we proceed to the details of the evaluation criteria, we briefly present a summary of the data sets used.

First, we discarded users if they did not have views during both the training and the testing periods. Our evaluation mechanism allows access to the training data for each of the algorithms, and restricts the algorithm to make recommendations to users based only on this information. If a user was not present during either of the periods, there is no way to either make any recommendation or evaluate the recommendations for that user. Similarly, unless a video was present in the training period, it cannot be recommended, and we removed such videos. If a video was present during the training period but not during the testing period (it

is possible that it had been removed from the system), we do not penalize any algorithm, since this case is easily handled outside the scope of the recommendation system, and is not a reflection of algorithm performance. After these operations, we kept approximately 1.1 million users and 1.3 million videos, with approximately 12.5 million watches.

Suppose a user watched a set W_1 of videos during the training period, and a set W of videos in the test period, and further suppose that the recommendation algorithm, after being given all data from the training period (for all users and videos), recommends a ranked list R of videos to the user. For $t = 1, \ldots, |R|$, let R_t denote the set consisting of the first t videos in R (in rank order). The following definitions are standard in information retrieval:

Precision at t, given by $precision_t(W, R) = |W \cap R_t|/|R_t|$, and

Recall at t, given by $recall_t(W, R) = |W \cap R_t|/|W|$.

Thus, for each user u who was present in both the training and testing periods, and legitimate value of t, we obtain a pair (p_t^u, r_t^u) of values[2]. Naturally, we may average these across all users to obtain a pair (p_t, r_t) of values for threshold t. These values form the basis of the in-depth analysis, including ROC curves (Receiver Operating Characteristic), Precision-Recall-Threshold curves, and top-rank quality assessment. We will also analyze the coverage over the set of videos by each recommendation procedure, see Section 4.

3.1 Backtesting Recommendations

We pause to critically analyze our evaluation methodology via backtesting of the kind outlined, point out the subtleties and caveats involved, and our rationale for adopting this method.

When rankings are not available, and the goal of a recommendation system is to simply entice the user to view the item of interest, the easiest method to evaluate a recommendation algorithm is to either run the system in live service and measure the number of recommendations that are viewed by users, or to have human raters manually evaluate the suggestions. However, both of these approaches have severe pragmatic difficulties. In our studies, many parameters settings and algorithm variations were explored; testing all of them in live service would have been impractical. The need for running each variation with enough users for a long enough time to obtain statistically significant differences conflicts with the need to ensure that all of our users get a consistent, good, experience on YouTube. On the other end of the spectrum, human rating cannot effectively scale due to the number of videos and algorithm variations attempted.

Perhaps one of the most interesting and challenging aspects of our study is the evaluation of the effectiveness of the suggestions *before allowing a live launch*. To evaluate the recommendation systems, we utilize historic log-data collected on user-video watches. Recall that to train the recommendations, we used anonymized log data for the first 46 days of the evaluation period. To test the recommendations, we employ similar log data from the latter half of the evaluation period.

A seemingly simple method to evaluate an algorithm's suggestions is to measure, for the video recommendations that would have been made by a system at time T, how many of the videos were actually seen by a user in the period $[T, \ldots, T+E]$. However, using historic log data for testing a recommendation system (whether it is adsorption, or based on any other technique) must be approached carefully. Some of the difficulties, and how they are addressed are listed below.

(1) Hindsight is not 20/20: If a suggestion is made to a user that the user did not view in the evaluation period, marking it incorrect is not necessarily appropriate. If the recommendation *had been* shown, the user *might* have watched it. The fact that it was not watched may simply be indicate that it was not discovered by the user.

(2) The number of videos watched in the evaluation period varies dramatically by user: in evaluating a recommendation system, evaluation is commonly done at certain settings (i.e. 100 recommendations presented). If the cases of users watching more/less than the recommended videos are not handled correctly, all evaluations may be artificially pessimistic.

(3) Recommendations are made at a single point in time: in this evaluation, every recommendation system is allowed to make its recommendations at the beginning of the evaluation period. Information during the evaluation period, such as new trends, video seen during the evaluation period, and emerging interests, are not allowed to be considered.

(4) New Videos: because of the rapid growth of the YouTube repository, many videos will be uploaded in the evaluation period that were not present in the training set.

(5) New Users: There will be new users who use YouTube for the first time to view a video during the evaluation period. These users will not have recommendations.

The easiest of the cases to handle are (4) New Videos and (5) New Users. For simplicity in the evaluation of all algorithms, we require that videos and users exist on YouTube during the training and testing phases of the recommendation systems in order to be counted in the evaluation. In deployment, this restriction will not be necessary; all models will be continuously updated. Note that this constant retraining also handles problem (3) that recommendations are made at a single point in time; new video watches and new popular videos will be taken into account. Despite these differences, the performance measurement at a single time point provides valuable information, and at worst underestimates the performance levels[3].

The problem (2) of users having different number of watched videos can be addressed in variety of ways, as illustrated in the following example. User U has watched two videos in the evaluation period; the Watch set W is $\{a, b\}$. Imagine that recommendation system R_1 is designed to recommend 5 videos, and it recommends (in the order shown) $\{a, c, b, e, f\}$. A second system, R_2, recommends (in the order shown) $\{a, c, e, f, b\}$. Let us attempt to evaluate which recommendation system is better for user U.

It is intuitively tempting to limit the size of the recommended list to at most $|W|$; if the user watches $|W|$ videos

[2]If R_t contains all of W (the procedure has recommended all of the user's views), then we drop user u in producing (p_s, r_s) for $s > t$, since including them artificially deflates the precision values without increasing the recall.

[3]Many of the issues associated with recommending new content are explicitly handled in YouTube through the separate sections for "New", "Popular" and "Recommended" videos. With respect to how soon new users will start receiving effective recommendations, as will be shown, this can start as soon as 2-3 videos are watched

then perhaps we should test our recommendations with $|W|$ videos. Note that with this size restriction, both R_1 and R_2 have the same precision and recall (a correct, b incorrect). If we do not limit the size of the recommended list to $|W|$, both have the same recall at 5 recommendations. However, the performance at the intermediate recommended-list sizes will differ. Therefore, when displaying the ROC curves for each algorithm, R_1 will appear better for the user than R_2 since it ranked U's second watch b higher than R_2 did. Because of this added insight into algorithm performance, we do not scale the size of the recommendations based on $|W|$ [4].

Finally, we need to address issue (1) of hindsight not giving perfect answers — even looking at historic logs, we do not know what action a user *would have* taken had a particular recommendation been made. Unfortunately, there is no way to overcome this problem; the precision/recall numbers that we obtain will not capture the full benefits of any system. Nonetheless, they provide meaningful numbers with which to compare the approaches relative to each other. They measure which approach better captures the *known* actions of the user — even when the user may not have had perfect information about all of the available videos.

In summary, backtesting provides a way to rank the expected performance of the suggestions system relative to each other, given that the users may not ever have full knowledge of the options (videos) available to them. Most importantly, this ranking can be conducted without subjecting users to multiple evaluation tests and the associated potentially inconsistent results.

3.2 Algorithms and Variants Tested

We briefly outline the three algorithms that we considered as good reference points for YouTube recommendations. While the literature is replete with recommendation algorithms, see for example [10], we have attempted to capture the fundamental insights of many of them and present a representative, easily analyzable, and functionally diverse, set.

The first, most basic, reference algorithm is called GP, for *global popularity*. This algorithm first sorts all videos by their popularity during the training period. When we need recommendations for a user u, we proceed in order of this popularity measure, and output top videos that the user has not watched during the training period. Overall popularity of a video is an extremely important piece of information in choosing videos to show users. As a reference, it ensure that any proposed algorithm gracefully handles the reality that for most users, many of their video watches correspond simply to globally popular videos (because they are often written about in blogs, linked to from emails, featured on the YouTube homepage, or are otherwise universally appealing).

The second algorithm, called LP (for *local popularity*) takes into account the fact that what is globally popular is poorly personalized for many users. The idea behind this algorithm is another very commonly employed item-based-recommendation heuristic, and is based on co-views. For each video v, compute the list $C(v)$ of videos z that it was co-viewed with, that is, users who watched v also watched z (during the training period). Furthermore, for each video v and video z co-viewed with v, assign a score $c(v, z)$ given by the number of users who co-viewed v and z. When we need recommendations for user u, look at the set $W(u)$ of all videos watched by u during the training period, and consider the set $C(W(u)) \doteq \{z \in C(v) \mid v \in W(u)\}$ of videos co-viewed with the videos watched by u. For each $z \in C(W(u))$, assign a score $c(u, z) \doteq \sum_{v \in W(u)} c(v, z)$; sort this set by total score, and output the first (or the first few) from this sorted list that user u has not watched already. This is also easy to implement, and captures a social phenomenon common in many popular recommendation systems (e.g., Amazon.com's "users who bought X also bought Y").

Even though LP is somewhat personalized for users, it still suffers from the drawback that it tends to be biased in favor of generally popular videos. For example, for a user who has watched videos of a certain soccer player, it might offer popular soccer videos rather than rarely watched videos of the soccer player that the user cares more about. The third and final reference algorithm rectifies this: rather than define the "co-view score" $c(v, z)$ between videos v and z to be the number of users who co-viewed v and z, we define $c(v, z)$ by

$c(v, z) = \frac{|U(v) \cap U(z)|}{|U(v) \cup U(z)|}$,

where $U(v)$ denotes the set of users who watched v during the training period. As before, we define $c(u, z)$ for user u and video z to be $\sum_{v \in W(u)} c(v, z)$. We call the resulting algorithm LR (for *local rare*). It is easy to see that this algorithm is biased against popular videos, and can be expected to produce highly personalized suggestions. The scoring formula is the standard Jaccard coefficient, a very commonly employed similarity function for two sets.

We compare two variants of the adsorption algorithm (with and without a dummy node) with the reference approaches. Both variants were implemented using the "averaging version" of the adsorption algorithm in the MapReduce programming model (Section 2.2) for 20 iterations, at which time the set of recommendations had largely converged.

ADSORPTION-N: At each user node, injection probability and dummy probability are 0; probability of continuing the random walk is 1. At each video node, injection probability is 1/4, dummy probability is 0, and probability of continuing the random walk is 1/2.

ADSORPTION-D: At each user node, injection probability is 0, dummy probability is 1/4, and probability of continuing the random walk is 3/4. At each video node, injection probability is 1/4, dummy probability is 1/4, and probability of continuing the random walk is 1/2.

Finally, we point out how the adsorption algorithm is used for the video recommendation problem. One way to use the algorithm is to run the algorithm on the user–video view graph, and use the label distribution derived for each user as the recommendations; the labels will be the most relevant videos for that user. An equivalent mechanism is one where for each user, we collect the distributions computed for each of her watched videos and take their average. Although this yields equivalent answers, it has an important benefit: we can use this procedure to make recommendation to new users (those that were not in the training set), as soon as they watch even *a single* video.

[4]Note that analogous arguments can be made for the case in which $|W| > |R|$. However, given the time spans used, this rarely happens in practice, and those cases do not change the overall results presented. Therefore, we also do make any special adjustments to handle this scenario.

Due to space restrictions, we do not delve into detail on the uses of the video–video co-view graph. However, a full set of experiments were conducted using this graph also; the results were very close to those presented with the user–video view graph presented in the next section.

4. EMPIRICAL RESULTS

We now summarize the performance of our algorithm on the data set harvested from YouTube logs. We ran each algorithm (the reference algorithms and the two variants of the adsorption algorithm) to produce a list of related videos for each video node, and for each user, up to 100 recommendations were made, ensuring that the algorithms never recommended videos already watched during the training period.

4.1 Reference Algorithms

We begin by presenting the ROC curves — (precision, recall) pairs obtained for various number of recommended videos — for the three reference algorithms, see Figure 3. Let us first note that points with higher precision and lower recall values correspond to picking the top few (unwatched) recommendations from the output of an algorithm (the right hand side of the figure), while points with higher recall and lower precision values (left hand side of the figure) correspond to using a large number of recommendations from the output of the algorithm.

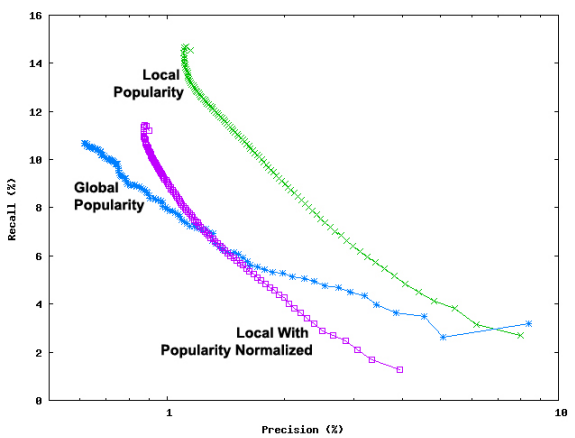

Figure 3: ROC for the Reference Algorithms

The first observation, one that is not at all obvious *a priori*, is that trends exist at collective and individual video watches by the YouTube user community. Considering the relative young age of YouTube, it would not have been surprising if most of the YouTube viewers were casual "one-off/occasional" viewers, for which anything other than the most simple models (always predicting the most popular) would not have yielded improvements. This, coupled with the fact that our evaluation mechanism is extremely stringent (across two 46-day periods), it is somewhat surprising that successful recommendations could be made at all.

One may wonder why overall, the precision values appear quite small. However, this value is deceptive for two reasons. One is that our evaluation is based on the backtesting paradigm, which is very conservative (it is possible that real users might have liked a recommendation if they had seen it, but that cannot be quantitatively evaluated). Secondly, even a precision value of 1% at top-100 is quite impressive, for this means that even for users with a very small number of watches in the test set, we manage to successfully produce at least one of their videos in the 100 recommendations we produced. Considering that the distribution of watch frequency is a typical power-law distribution with mean roughly 5, with most users watching below average, this is quite surprising.

With its top one recommendation, Algorithm GP achieves roughly 8.4% precision at 3.2% recall, and with its top 100 recommendations, a recall of 10.6% at a precision of 0.6%. The success of this simple heuristic with just one recommendation is quite impressive, and sets a strong baseline for our algorithms to measure against.

The next interesting phenomenon we observe from this plot is the cross-over of the ROC curves for GP and LR, which achieves higher recall at lower values of precision. This suggests that LR can tailor results better for each user, when it is allowed to make a large number of recommendations. The conclusion is that if we would like to make a small number of recommendations, it is better to make suggestions that are globally popular; and if we are allowed to make a larger number of recommendations (over time, say), there is opportunity to take full advantage of co-view statistics, an encouraging trend.

Finally, LP, which effectively combines these two, dominates both of these reference algorithms nearly uniformly (except at top-1 position, see Section 4.3), and the gains are significant.

4.2 Adsorption Algorithms

We now turn to the performance of the adsorption algorithms. For simplicity, we restrict the comparison to LP, since it is clearly the strongest of the reference algorithms across nearly the entire spectrum of trade-offs. The ROC curves for the two variants of the adsorption algorithm, along with that of LP, are presented in Figure 4.

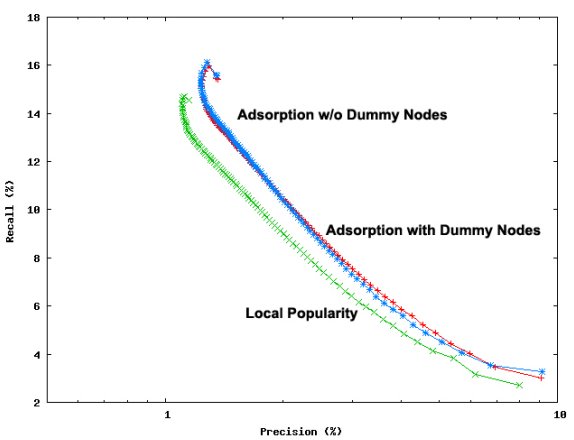

Figure 4: ROC for the Adsorption Algorithms

First, we note that the performance of ADSORPTION-D and ADSORPTION-N are nearly identical, with slight gains for the former in the higher precision regime (fewer recommendations) and slight gains for ADSORPTION-N in the higher recall regime (more recommendations). This is consistent with our general intuition that hobbling a random

walk keeps it closer to the source node, so there is a smaller chance of topic drift, and hence higher precision. Conversely, a less inhibited random walk reaches farther, and is able to achieve a higher recall. The one exception to this general rule is at position 1, where ADSORPTION-N has better recall than ADSORPTION-D. However, as we shall see shortly (in the analysis of Figure 5), the picture is more intricate than would seem at first.

The second clear conclusion from Figure 4 is that at all operating regions, the adsorption algorithms perform better than the best baseline. For most values of precision, we observe a gain of 17% in recall over LP, and for most values of recall, we observe 21% improvement in precision over LP. Gains of these magnitudes often translate into several tens or hundreds of millions of clicks on a system of the scale of YouTube, and are very valuable.

4.3 Top-1 performance

In recommendation systems, it is natural to expect that the more information the system has about a user, the better targeted the recommendations will be. We seek to quantitatively understand the influence of the users' viewing frequency on the performance of our algorithms. To keep the discussion rooted to one control parameter (view frequency), we focus on the performance of the algorithms with respect to their top recommendation for each user. As we shall see, this turns out to be an excellent microcosm of the complex relative behavior of the various algorithms.

When we fix a user, and ask each algorithm to make one recommendation to this user (that they have not already watched in the training period), the single bit of information that determines both precision and recall is whether the user watched this video during the test period. Therefore, we will plot the average of this Boolean variable across all users with a given number of views in the training period. For visual simplicity, the number of views is rounded to the nearest power of 10 (so it corresponds to a precisely marked point in log-scale), and simple smoothing of the graph is performed; the results are shown in Figure 5.

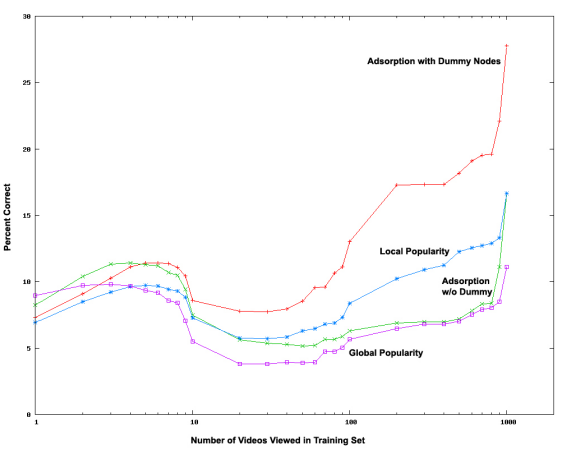

Figure 5: Precision/Recall at Rank 1

A number of interesting facts emerge from this plot. For convenience of exposition, we will denote the x-axis parameter of this plot consistently as *view frequency*.

First, note that GP has the best performance for users who viewed just one video during the training period, but it quickly degrades as the view frequency increases, becoming the worst among all the algorithms! This suggests that for very casual users (who visit a system infrequently), recommending the most popular videos is a useful strategy. However, for users with very high view frequency, the idea of the "popular but unwatched" videos is a somewhat questionable notion (since if something is globally very popular yet they haven't watched it, it suggests that they are likely not interested in that genre/topic). Importantly, it also suggests that as the amount of information about a user increases, and we are given more chances to recommend videos, we can do much better than simply recommending the most popular videos.

Algorithm ADSORPTION-N has a behavior similar to GP, and has the best performance among all algorithms for small values of view frequency. We shall say more on this shortly.

Algorithm LP quite strongly takes into account what the users have seen, and outperforms GP for users with as few as 5 views. At this range, this algorithm tends to recommend popular videos that are related to what the user has watched, for example, popular ones in the same genre (music, sports, etc.) that the user has exhibited an interest in. The success of this algorithm continues as the view frequency increases, but is somewhat modest. On the other hand, it should also be noted that LP is quite poor for users with a very small number of views (nearly 20–25% worse than GP).

Algorithm ADSORPTION-D performs significantly better as the view frequency increases beyond 5, and the gains become much larger as the view frequency increases further (nearly 50% better than the nearest competitor at view frequency 100, and over 30% better at view frequency 10). This algorithm also handles the phase transition from popular recommendations (view frequency under 5) to personalized recommendations (view frequency between 10 and 100) better than the others in that it quickly recovers its peak performance and continues to increase strongly beyond that.

The other interesting fact that emerges from Figure 5 concerns the contrast between the adsorption algorithms with and without dummy label injection. Going back to Figure 4, we saw that ADSORPTION-N performs better than ADSORPTION-D with respect to the top recommendation (highest precision point); what is missing from the aggregate statistic there is the fact that the relative performance between these two algorithms, even for their top recommendation, depends crucially on what type of user we consider, as is clear from Figure 5. Most users watch a small number of videos, so the overall precision-recall point for the algorithms for their top recommendation depends heavily on the behavior at the lower values of view frequency. In this regime, algorithms that recommend globally popular videos tend to do better, but for the heavy tail of the user base — the significant number of users who watch much more beyond the casual user (and the set of users for whom recommendations actually matter) — the more refined algorithm ADSORPTION-D has tremendous advantages.

A plot of the form Figure 5 may be considered a "guidance plot" to select among algorithms depending on individual users' usage pattern. In the machine learning literature many methods of combining different models have been proposed [4]. This plot makes selecting between models a trivial process, one based on an easily measurable and intuitive number: the number of videos a user has seen in the training

period. The more videos seen, the more confidence we have in suggesting less common, more specialized-interest videos.

There is another interesting insight into the behavior of the adsorption algorithm that we may glean from Figure 5, and this concerns the reason *why* algorithm ADSORPTION-N recommends popular videos for users with a small number of views, compared to ADSORPTION-D. Recall that ADSORPTION-N performs a less constrained random walk than ADSORPTION-D so it reaches a wider set of video nodes — however, since we only maintain the top 100 labels at each node, the only ones that survive are those that are encountered frequently in the random walk, in other words, the popular ones in the vicinity of the user node. Algorithm ADSORPTION-D, by contrast, prunes several paths in the walk, so is able to explore further along the more relevant paths, leading to less popular videos. In conclusion, while recommending the most popular available choice makes sense for users with just one view, anything beyond one view allows us to do better than recommending the most popular video (as evidenced by ADSORPTION-N, which does better than GP for view frequency 2).

Finally, Figure 6 plots precision and recall for ADSORPTION-D as a function of the number of recommendations.

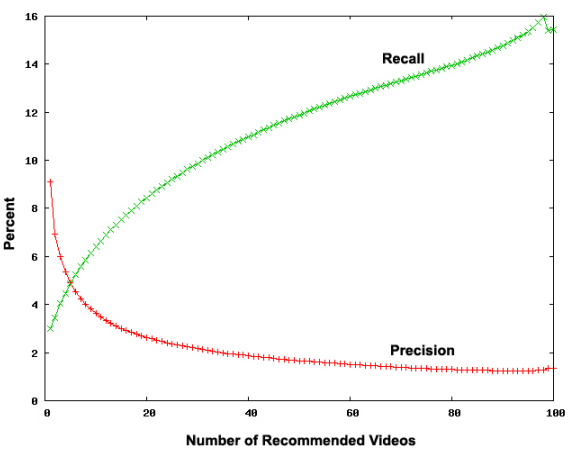

Figure 6: PRT for Adsorption Algorithm with Dummy

Finally, due to space constraints, we omit discussion of the coverage of the algorithm, namely how much of the set of videos each of the algorithms has in its bag of all recommendations. These are available in the full version of this paper, available from the authors.

5. CONCLUSIONS

By using the adsorption algorithm, we were able to improve the expected efficacy of suggestions in YouTube. The most commonly used heuristics, recommending the overall most popular videos and/or the most co-watched videos did not perform as well. Because of the short half-life of videos on YouTube, the large number of uploads, and the exposure that users have to new and popular videos, recommending anything other than commonly viewed videos was not guaranteed to provide improvement. However, the fact that trends can be found provides strong evidence not only in favor of a graph-based algorithm, but that there is indeed interesting usage information to be mined from YouTube beyond the casual viewing of popular videos. Additionally, we presented a method by which to backtest recommendation systems through historical log-analysis.

In standard machine learning parlance, we have presented a simple algorithm that is able to use an underlying graph, which contains potentially noisy data, unlabeled nodes, and non-uniform connectivity, to propagate a very large number of labels to each node. We are currently exploring the use of this procedure in numerous domains, including advertiser targeting, product recommendations, labeling web-images, and detecting threads and story-lines in news stories.

6. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE-TKDE*, 17(6), 2005.

[2] A. Azran. The rendezvous algorithm: Multiclass semi-supervised learning with markov random walks. In *ICML-24*, 2007.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7), 1998.

[4] K. Burnham and D. Anderson. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer-Verlag, 2 edition, 2002.

[5] L. Candillier, F. Meyer, and M. Boulle. Comparing state-of-the-art collaborative filtering systems. *ML and DM in Pat. Recog*, 2007.

[6] China Post. Youtube launches site in traditional Chinese `http://www.chinapost.com.tw/`.

[7] G. Cormode, F. Korn, S. Muthukrishnan, and Y. Wu. On signatures for communication graphs. In *Proc. 24th IEEE-ICDE*, 2008.

[8] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. 6th SOSP*, 2004.

[9] P. Indyk and J. Matousek. Low distortion embeddings of finite metric spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Disc. & Comp. Geom.* 2003.

[10] KDD Cup 2007. Who rated what in the netflix challenge. `http://www.cs.uic.edu/ liub/netflix-kdd-cup-2007.html`.

[11] NetFlix. http://www.netflixprize.com/.

[12] C. Snoek and M. Worring. Multimodal video indexing: A review of the state of the art. *JMTA*, 25:5–35, 2005.

[13] M. Szummer and T. Jaakkola. Partially labeled classification with Markov random walks. In *NIPS*, 2001.

[14] YouTube. `http://www.youtube.com/results?search_query=*&search=Search`.

[15] X. Zhu. *Semi-Supervised Learning with Graphs*. PhD thesis, Carnegie Mellon University, 2005. PhD Thesis.

[16] X. Zhu, G. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *ICML-20*, 2003.