Recommending Related YouTube Videos

Shantanu Thakoor, Anchit Gupta, Parth Shah
{thakoor, anchitg, parth95}@stanford.edu

Abstract

Building a recommendation system for YouTube represents a problem of large scale and huge importance. In this paper we explore a recommendation system, which unlike previous approaches more directly relies on YouTube's inherent graph structure. Using graph node embeddings we train a neural network for link prediction and subsequently tackle the challenge of scalability by combining this with a locality sensitive hashing based approach. We also show how our approach extends to provide recommendation given a set of user rated videos.

1 Introduction

Youtube is the world's largest online video content provider. It is responsible for serving over 5 billion videos every single day to a total user base of size 1.3 billion. Building a good quality recommendation engine to serve this enormous group of people and help them sift through the billions of minutes of videos is of huge importance.

In this project, we seek to tackle the problem of YouTube Video Recommendations. Specifically, given a set of videos and their ratings, we wish to suggest other videos similar to the well-rated videos and dissimilar to the badly-rated videos - this could be used, for example, to suggest a new video to a user based on videos they have already liked, or to suggest a suitable additional video for a play-list a user has created.

Traditionally this problem has been solved by using an approach based on classical recommender systems where we are trying to recommend an item to a user based on their preferences. Traditional recommender systems use techniques like item based collaborative filtering [ASA+10], which use notions of distances between items. Methods like this rely on having utility matrices available and cannot scale well to the huge sizes involved in YouTube. Also these distances are calculated using feature vectors of items, and do not use any graph information.

Our approach relies on using graph information to give better recommendations and also a much more scalable system. We cast YouTube recommendation as a network problem, where given a set of nodes we wish to predict other nodes that are closely related to those in our set. The bulk of our approach relies on using *node2vec* [GL16], where nodes are embedded into a low-dimensional vector space that has certain nice properties of distances between neighbors and graph structure being preserved. After computing these vectors for our graph we proceed to train a neural network for link prediction on these vectors.

To make our approach more scalable we subsequently do locality sensitive hashing of these vectors for scalable nearest neighbor queries. We also investigate combining the neural network based link predictor with the hashing based nearest neighbor search to further filter out the videos and obtain good quality recommendations.

The rest of the paper is organized as follows. In Section 2, we give a brief overview of past work tackling similar problems. In Section 3, we describe the dataset we use, and conduct preliminary analysis of the dataset with experiments relevant to our task. In Section 4, we begin addressing our main problem, by applying the *node2vec* technique to our dataset. We also present analysis on its

efficiency and our ability to predict edges based on it. In Section 5, we further develop our main approach, by presenting our idea of using Locality Sensitive Hashing to provide a more scalable method, and commenting on its effectiveness. Finally, in Section 6, we provide a brief summary of our work, insights we gain from it, and possible future extensions.

2 Related Work

Extensive work has been done on recommendation systems based on per user item utility functions. These can be broadly classified into three categories 1) Content based 2) User collaborative 3) Item collaborative . For a more detailed description, we refer the reader to [ASA+10].

Using neighborhood information for collaborative filtering recommender systems has previously been studied in [Dom07]. The authors in [SP14] propose a scalable peer to peer recommendation system based on distributed hash tables and locality sensitive hashing of user-item preference matrices.

Based on ideas inspired from the above, [DLL+10] build a YouTube recommendation system by computing candidate videos based on relatedness scores between videos and then a ranking algorithm. More recently [CAS16] propose a algorithm having both candidate generation and ranking done using deep neural networks. These methods differ from our approach as they use complete user watch and click data unlike our more restricted dataset and also they do not incorporate graph information directly into their models.

The graph based approach we adopt can also be seen in the light of link prediction. These methods mainly rely on graph heuristics based on graph properties like common neighbors, path lengths and neighborhood sets.[LNK03] provides good background on most of the classical approaches for the same.

Recently *node2vec* [GL16], a node embedding approach based on random walks, has shown much better accuracy than the above methods for link prediction and we use the same in our work. In Section 4, we provide a more detailed explanation of the *node2vec* method. [RY17]looks at making a Reddit recommender system based on text data gathered from user comments and incorporate the graph structure of the social network into their system using *node2vec*.

For a more detailed treatment of the above papers, we refer the reader to our project proposal document.

3 Analysis of Dataset

3.1 Dataset Description

We use the YouTube dataset found at [CDL08]. The dataset contains video IDs, along with some metadata including uploader, length, ratings, category, age, and a list of up to 20 IDs of related videos. The dataset was created by starting a breadth-first search crawl starting from videos in YouTube's set of Most Viewed, Top Rated, Recommended, and other playlists. The dataset contains information from more than 60 such crawls, most of which discovered videos on the order of 10^6 to 10^7. Due to scalability issues, in this analysis, we restrict our attention to one particular crawl over YouTube videos.

3.2 Working with Maximum Strongly Connected Component

We analyze the natural graph representation of this dataset, with nodes representing videos and an edge from a to b meaning that b was in the list of related videos for b. Our graph contains over 1.76M nodes, and 4.4M edges. In Table 1 we examine the out-degrees of nodes in the entire graph, and in the largest strongly connected component.

We can see that in the entire graph a large majority of the nodes have out degree 0 and are sink nodes. For our approach, sink nodes are essentially nodes for which we have no information. Hence, *node2vec* (which finds related nodes using random walks starting at each node) will not be able to produce vectors that can easily be used in downstream tasks. Since the vast majority of nodes in the original graph have little to no information associated with them, we restrict our atten-

tion to the SCC. In Section 4, we will see experimental results showing that *node2vec* indeed does not give good results for the entire graph, due to lack of information associated with the majority of the nodes.

Further, we also examine the clustering coefficients for the entire graph (0.142) and the SCC (0.441). For our problem, the clustering coefficients are a rough indicator of the difficulty of our problem - if the clustering coefficient is high, our problem is in some sense easy, since the natural algorithm of giving prediction as a neighbor's neighbor (ie, a node 2 hops away) will work well. On the other hand, a low clustering coefficient would mean that there is not much benefit to using graph information in our recommender system. Hence, this is a further motivation for working only with the SCC - we have reason to believe that our methods would be more applicable in this setting.

Finally, consider the way this graph has been generated - using a Breadth First Search crawl from a node, with us discovering at most 20 new nodes at each step. Now, the majority of the nodes will be discovered in the last step of the crawl, due to the exponentially growing nature of the BFS tree. These nodes will not have any outgoing edges, since the BFS has ended. Hence, our dataset is only meaningful when we consider the SCC, where each node present has had outgoing edges explored.

Out Degree	Proportion of nodes in the entire graph	Proportion of nodes in SCC
0	0.868	0
$1 \leq \ \leq 5$	0.0004	0.337
$6 \leq \ \leq 10$	0.0006	0.327
$11 \leq \ \leq 15$	0.0178	0.218
$16 \leq \ \leq 20$	0.113	0.118

Table 1: out-degree distribution

3.3 Analysis of SCC

The SCC has over 216k nodes and 1.85M edges. Hence, focusing on a subset of the graph does not render our problem trivial, since the size is still quite large. To examine the structure of the graph, in Figure 1 we plot the in-degree distributions of the graph. The in-degree of a node is somewhat analogous to the popularity of the video; the more popular a video is, the more videos will be linking to it, and hence the more it's in-degree would be.

We observe that the in-degree behaves as expected - there are few very popular videos of high in-degree, and many videos that have only a few others linking to them. We can make the observation that a good null model graph for this dataset would be the preferential attachment model [BA99] - since it is likelier for new videos to link to already popular ones, rather than to link to unpopular ones. Hence this "rich get richer" type of graph formation gives rise to the degree distribution observed.

Next, we examine the metadata of the videos to see whether the features provided can themselves form a good baseline for prediction. One natural approach is to group videos based on their category, since we would expect related videos are always in the same category. In Figure 2, we plot a histogram of clustering coefficients of the various subgraphs of the SCC that belong to the respecting categories, and compare with clustering coefficient of the overall graph. As explained earlier, we use the clustering coefficient as a proxy for measuring ease of prediction. As we can see, the clustering coefficients are much lower - which would not be the case if edges were often in the same category. This may be explained as the YouTube algorithm balancing exploration and exploitation, and trying to present the viewer with a diverse set of recommendations, as explained in [DLL+10]. We measure the probability of a recommended video being of the same category as $\boxed{p = 0.5958}$ Another natural idea is to predict other videos by the same uploader. We measure the empirical probability that an edge connects videos created by the same uploader, and find that the probability $\boxed{p = 0.1590}$

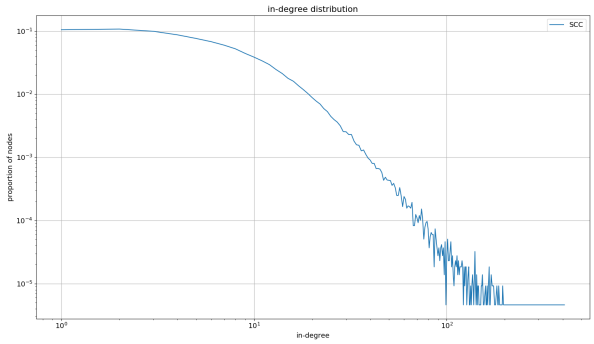


Figure 1: In-degree distribution of the SCC graph

These two experiments show that any naive method of recommending based only on the feature vectors is unlikely to result in a good performance, since the category and uploader are by far the most expressive features and do not give good results. Hence, our motivation to use graph information in addition to just the features will likely result in a better performance. In Section 4, we explore how to create easy to use features from the graph information using *node2vec*.

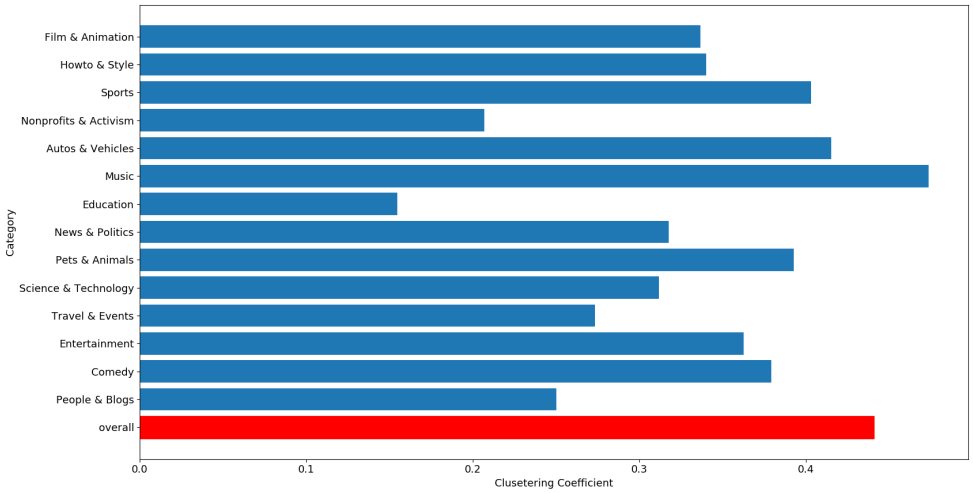
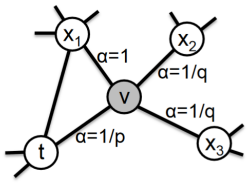


Figure 2: Clustering Coefficient for individual categories

4 Vector Embeddings of Graph Nodes

node2vec [GL16] allows us to learn fixed-length feature representations of nodes in a graph. It learns a mapping from the set of nodes to a lower dimensional real vector space $\mathbb{R}^d$ such that the likelihood of recovering the neighbors of a node from these vectors is maximized. Their key insight lies in the ability to define a objective function optimizing which gives us the vector representations. They use a set of biased random-walks to generate the neighborhood of a node. These can be tuned using parameters which defines a notion of a neighborhood for our particular graph. Consider a walk that has just traversed (t, v) and is now at v, it transitions using a edge (v, x) with probability $\pi_{vx} = \alpha(t, x)w_{vx}$ where w_{vx} is the edge weight and

$$\alpha(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$



d_{tx} is the length of the shortest path between t and x. The parameters p and q can be tweaked to approximately interpolates between a Depth First and Breadth First search. Using these generated neighborhoods we are now able to compute the gradients for out objective function and use Stochastic Gradient Descent [Bot10] to optimize it.

As the above model is trained to predict neighborhoods it is particularly useful in our setting where we would like to predict related videos which might not have been connected by a edge already.

4.1 Analysis of *node2vec* on the Youtube Data Graph

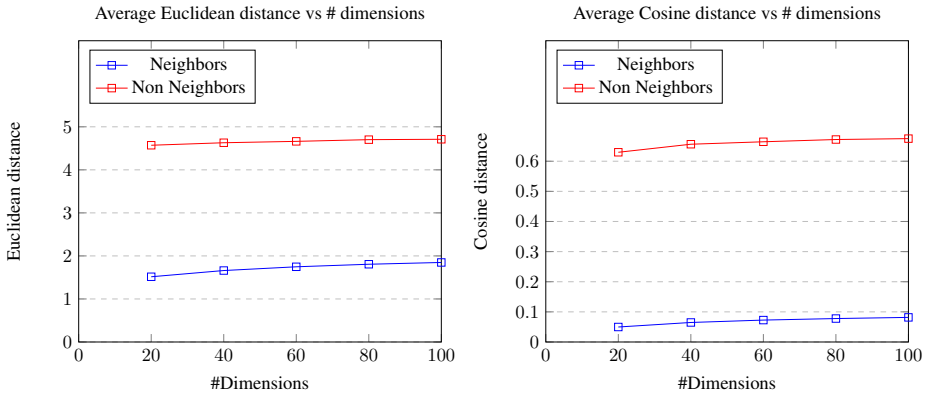
We run the *node2vec* algorithm on our dataset and analyze the embeddings produced. We first ran *node2vec* on the whole unmodified graph to get 60 dimensional vectors and computed the average Euclidean distance between neighbors to be $\boxed{1.23}$. We then estimated the average distance between non-neighbors by sampling some random nodes, which surprisingly came out to be $\boxed{0.9482}$. Since the average distance between neighbors was higher than the average distance between non-neighbors, these learned embeddings clearly did not preserve distances well upon projection from the original metric space.

As described in the previous section, around 90% of the nodes in our dataset are sink nodes. *node2vec* uses random walks starting from each node, to explore the graph structure. As random walks from these sink nodes are empty, hence *node2vec* is not able to compute satisfactory embeddings for such a graph.

Hence to combat this problem we only work with the strongly connected component of the graph. We subsequently analyze these embeddings in detail.

4.2 Analysis of *node2vec* on the SCC

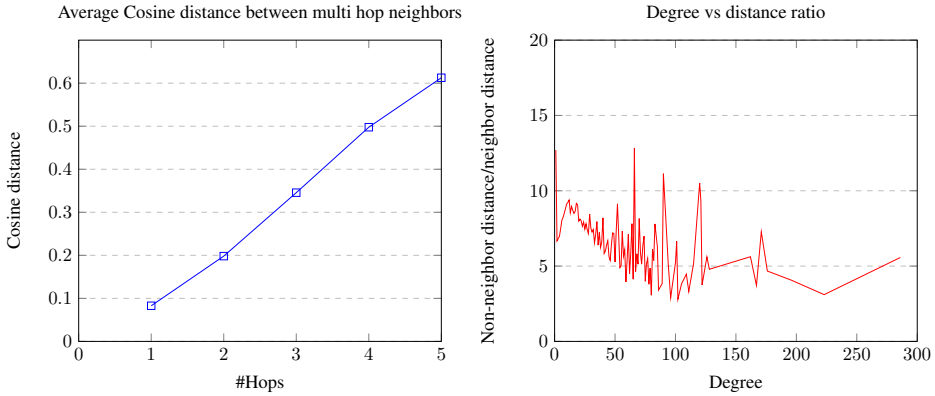
To try and find the optimal number of dimensions to use, we plot the average distance between neighbors, and the estimate of the average distance between non-neighbors, as a function of the number of dimensions in our learned embeddings.



As is evident from the above graphs, there is not much difference in distances with change in dimensions. *node2vec* seems robust enough to capture neighbor distances even with low number of dimensions. We find that in downstream tasks where we use the vectors for non-trivial recommendations, the difference between number of dimensions is more evident and hence we set the number of dimensions to 60 for a good balance between fast computability and performance.

Also, note the difference between the euclidean distance and cosine similarity - there seems to be a larger gap between the two distance plots when using cosine similarity. This suggests that cosine

similarity based on angular difference is a better metric than euclidean distance, for the link prediction task.



Next, we plot the average cosine distances between k-hop neighbors while varying k. This gives us an idea of how well *node2vec* is able to preserve neighborhoods in the original graph. As is expected, the distance increases linearly with the number of hops.

Another plot of interest is seeing how the ratio of average neighbor vs non-neighbor distances changes for different degree nodes. Due to computational constraints, this data was gathered by randomly sampling nodes. There is an evident decreasing trend in the ratio as the degree increases. This can be explained by the fact that high degree nodes have a large set of nodes that can be reached with 2 hops, a even larger set reachable with 3 hops and so on. Hence these high-degree nodes have a "central" location in the graph and are close to a big set of nodes. Our *node2vec* embedding seems to capture this structure as well.

In conclusion, *node2vec* does a very good job of capturing the network properties in our graph. Hence, we are justified in using it as a proxy for network structure in our recommendation tasks.

4.3 Link Prediction Using *node2vec*

To generate recommendations and predict new edges, we frame the problem as a supervised learning instance - specifically, as a classification task, where we wish to predict whether there exists an edge between a pair of nodes. Given two videos u, v and their corresponding to vector embeddings $f(u), f(v)$ we compute the vector $f(u) - f(v)$ as input to our classifier. We also appended the cosine distance between $f(u), f(v)$ to the input based on our analysis of the embeddings in the previous section. As the complete dataset contained a huge amount of edges (over 1.8 million) and due to a lack computing resources, it was infeasible to train over all node pairs. Hence, we created a training dataset by randomly sampling edges from the graph. We also added an equal number of randomly sampled non-edge vertex pairs to the dataset, to remove bias from our model.

The first approach we tried was using a simple Logistic Regression model. Even after augmenting the dataset with some additional features like Euclidean distance, this was unable to converge to a useful solution, with accuracy hovering around 50%.

This suggests the underlying function to be learned is a lot more complex than a linear model could learn. Hence, we resort to training a neural network. The neural network we train has an input layer of size 61, and 2 fully connected hidden layers of size 120 and 30. The output layer was a softmax layer of size 2, since our task was binary classification. Between all the layers, we used batch normalization to speed up training. Our trained network has an impressive accuracy of 97.3% on an independently drawn test set. Our results are summarized in Table 4.3.

Model	Accuracy
Logistic Regression (baseline)	51.3%
Neural Network without cosine distance	83.1%
Neural Network with cosine distance	97.3%

Table 2: Link Prediction Accuracies

In our network, nodes having an edge is a proxy for the underlying videos being related. Hence, we have obtained a classifier that, given a pair of video, can very accurately predict whether they are related. In theory, this classifier could be used to find all related videos of a given node. However, doing so would involve iterating through all the other nodes in the graph. This is not feasible for large datasets such as ours. Instead, we plan to use a fast, approximate method to generate a list of nodes which are likely candidates for being related videos, and then use this neural network to further filter this list. In Section 5, we describe a Locality Sensitive Hashing based nearest-neighbor approach, to generate this list of candidates, and further extend this classifier which only predicts whether a pair of nodes is related, to a system which recommends a new video given a set of videos and their ratings.

5 Locality Sensitive Hashing for Approximate Related Video Prediction

In this section, we first provide motivation for using LSH as a scalable, approximate solution to our problem. Then, we conduct experiments showing that we can indeed find a hash function satisfying our desired property of locality preservation. Finally, we evaluate this approach on the task of giving video recommendations based on a set of videos and their ratings.

5.1 LSH as a Scalable Recommendation Algorithm

As detailed in [ASA+10], most recommender systems use a user-item utility matrix, forming a rich model of user preferences and item similarity. These approaches do not scale well - in particular, for the YouTube platform where there are millions of users and videos, it would be infeasible to store this information. However, we can assume that we have a limited amount of information about a particular user's preferences. Hence, given a set of videos a user has watched, and their ratings, we wish to provide the user with recommendations of new videos.

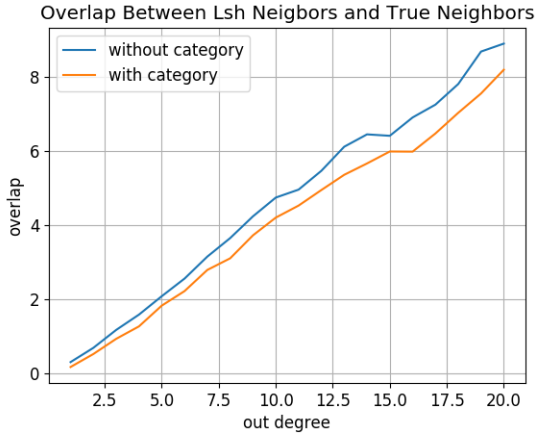
To solve this problem, we first recall the original work on which *node2vec* was based - the Word2Vec model introduced in [MCCD13]. This model embeds words into a low-dimensional vector space, for use in downstream NLP tasks. The *node2vec* model is quite analogous to Word2Vec in how it is trained and used. Later works in NLP considered finding an efficient and expressive representation of sentences. In particular, [ZKFM10] suggests finding a sentence vector by taking a linear combination of the vector representations of the words it contains.

We extend this approach, to representing a set of videos and their ratings, by taking a linear combination of the vector representations of the videos, using weights based on their ratings.

$$v' = \frac{\sum_i w_i v_i}{\sum_i w_i}$$

We use this weighted vector v' and find videos in its neighborhood. The intuition is that we will be finding videos close to the well-rated videos, and in a sense far away from the badly-rated videos. Hence, we can use efficient nearest-neighbor algorithms to find the best videos, and thus make our recommendation.

However, we face a further problem of scalability: nearest neighbor searches in a high-dimensional dataset are expensive. Traditional recommender systems can deal with this by taking advantage of certain properties of the utility matrix such as sparsity; since we do not have a full user-item utility matrix, we instead propose an efficient approximate solution. As detailed in [WSSJ14], can use Locality Sensitive Hashing, to find a single number that represents each node. We then conduct a nearest-neighbor search on this one-dimensional space. Since our hash function is locality-preserving, a nearest-neighbor search in this space would be a good approximate solution to a nearest-neighbor search in the original space. Such a one-dimensional search may also be implemented using a distributed hash table similar to the work by [SMK+01] - hence, this approach would truly be scalable for a dataset the size of YouTube.



5.2 Analysis of the LSH Results

We use the Locality Preserving Hash function provided by the scikit-learn LSHForest module [1]. We wish to verify that this hash function is indeed locality-preserving. To this end, we conducted the following experiment: for a node of degree d, we calculated the $1.5d$ nearest neighbors, the extra $0.5d$ being a fudge factor to allow for the approximate nature of the LSH function. We then calculated the overlap between the node's d true neighbors, and the $1.5d$ neighbors predicted by our algorithm. We calculated this both using the video category feature as part of the input to the LSH, and without. Finally, we plotted this function as amount of overlap as a function of the node degree in Figure 5.2.

As we can see from the figure, the LSH is quite a reasonable approximate method for finding the neighbors of a node. For nodes of all degrees, it predicts around half of their true neighbors. An interesting fact is that adding category information actually worsens our results - but this is not surprising, considering that in Section 3 we found that video category is not a reliable feature for edge prediction.

As a final experiment, given a node and its list of neighbors predicted by the LSH algorithm, we fed them to our classifier (as detailed at the end of Section 4). We obtained very good results from this experiment - on average, $\boxed{82.1\%}$ of the neighbors predicted by the LSH algorithm, were classified as having edges to the test node. In other words, the LSH algorithm predicts related videos with 82.1% accuracy. Hence, our LSH method is indeed an efficient yet accurate method for link prediction.

5.3 Experimental results based on a set of rated videos

The approaches detailed in prior sections solved the problem of predicting related videos for a single given video. We now extend this to predicting related videos for a set of videos and their ratings.

Since our dataset is almost 10 years old, most of the videos discovered in its crawls are no longer available on YouTube. We hence find the 500 top viewed videos that are still available by scraping the YouTube website, and use them to gain some insight into the data.

Since our dataset contains edges between pairs of nodes, we had data regarding single-video recommendations. Hence, we are able to present quantitative analysis in prior sections. However, we have no data regarding recommendations from a set of videos, and hence quantitative analysis is not possible. Hence, we present qualitative analysis to argue that our method gives reasonable results. The results are summarized in Table 5.3.

As a first test, we start with a seed set of 5 videos, all in the sports category and uploaded by the NBA channel. We take all the ratings (i.e, the weights in our linear combination) as equal. The 5

[1] scikit-learn.org/stable/modules/generated/sklearn.neighbors.LSHForest.html

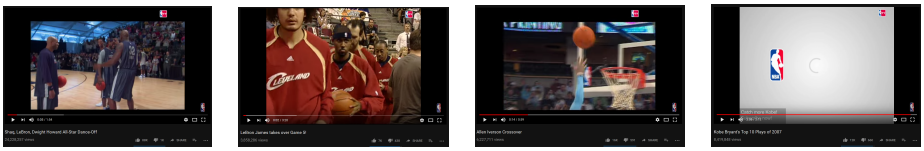


Figure 3: Seed set for prediction

Figure 4: Recommended videos

Type of Test	Seed Set	Result Set
Same category and uploader, unweighted	5 Sports videos from NBA uploader	5 Sports videos from basketball-related channels
Same category, different uploader, unweighted	5 Music videos from different uploaders	5 videos from Music and Entertainment
Different category, weighted	5 Music videos, highly rated; 2 Sports videos, low rated	4 Music videos, 1 Sports video
Same category, negative weights	3 positive rated Music videos, 3 negative rated Music videos from uploader UMG	5 Music and Entertainment videos, not from uploader UMG

Table 3: Results of Video Recommendation Using LSH

closest neighbors in the LSH space include videos in the sports category, mostly being uploaded by NBA, but also other basketball-related channels. The thumbnails of the corresponding seed set videos can be viewed in 3 and our recommendations in 4, you can zoom in to be view the video titles and other details.

Next, we relax the condition that the seed set contains videos by the same uploader. We start with 5 videos in the Music category but all different uploaders, and our 5 closest neighbors in LSH space are all in the Music or Entertainment category, with different uploaders.

Further, we relax the condition that all ratings are the same, by starting with a seed set of 7 Music and Sports videos, with the Music videos having higher weights. Our 5 closest neighbors in LSH space contain 4 Music videos and 1 Sports video.

Finally, we explore the case where some videos have negative ratings. We start with a seed set containing positively weighted Music videos of different uploaders, and negatively weighted Music videos by the Universal Music Group uploader. Our results contain videos in the Music and Entertainment category, none of which are from the Universal Music Group uploader. This uploader is one of the most highly present uploaders in the Music category, with over 900 videos, and hence not including any of their videos in our closest neighbors indicates that our approach is effective.

In all our experiments, we obtain qualitative results that are in line with what we would expect from a YouTube video recommender system. Hence, we believe that our method shows great promise and solves our problem effectively.

6 Conclusion

We explore the problem of YouTube video recommendations using graph information. After a thorough analysis of our dataset and its properties, we identify key structure in our problem. We

also realize that the video features provided by our dataset are insufficient for our task, and thus motivate using graph information.

We present a *node2vec* based method for link prediction, and demonstrate its effectiveness by conducting experiments where it performs to a high degree of accuracy. We then extend our method for single video recommendation, to a highly scalable Locality Sensitive Hashing based approach. We demonstrate that our fast, approximate method gives very high accuracies on the task of single-video recommendation. We also provide qualitative analysis and examples indicating that our method extends to making predictions from a set of rated videos as well.

Future work would include further using graph information and the video features together, to build a more accurate recommender system. It would also be interested to construct a dataset containing popular YouTube playlists and how videos get added to them. This would help us gain insight into our problem and provide a test dataset where we may quantitatively evaluate our approach as well. Finally, it would be interesting to run our experiments on the entire YouTube dataset, to measure the robustness and scalability of our approach on such massive amounts of data.

References

[ASA+10] Dhoha Almazro, Ghadeer Shahatah, Lamia Albdulkarim, Mona Kherees, Romy Martinez, and William Nzoukou. A survey paper on recommender systems. *CoRR*, abs/1006.5278, 2010.

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[Bot10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[CAS16] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[CDL08] Xu Cheng, Cameron Dale, and Jiangchuan Liu. Statistics and social network of youtube videos, 07 2008.

[DLL+10] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 293–296, New York, NY, USA, 2010. ACM.

[Dom07] Ramesh Dommeti. Neighborhood based methods for collaborative filtering, 2007.

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[LNK03] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, pages 556–559, New York, NY, USA, 2003. ACM.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[RY17] Xin Li Rex Ying, Yuanfang Li. Graphnet: Recommendation system based on language and network structure, 2017.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[SP14] A. Smirnov and A. Ponomarev. A hybrid peer-to-peer recommendation system architecture based on locality-sensitive hashing. In *Proceedings of 15th Conference of Open Innovations Association FRUCT*, pages 119–125, April 2014.

[WSSJ14] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014.

[ZKFM10] Fabio Massimo Zanzotto, Ioannis Korkontzelos, Francesca Fallucchi, and Suresh Manandhar. Estimating linear models for compositional distributional semantics. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 1263–1271, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.