Grale: Designing Networks for Graph Learning

Jonathan Halcrow†, Alexandru Moşoi‡, Sam Ruth†, Bryan Perozzi†
†: Google Research
‡: YouTube
{halcrow,mosoi,samruth}@google.com,bperozzi@acm.org

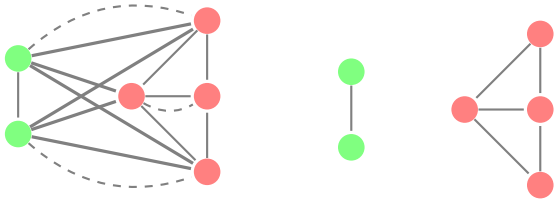
ABSTRACT

How can we find the right graph for semi-supervised learning? In real world applications, the choice of which edges to use for computation is the first step in any graph learning process. Interestingly, there are often many types of similarity available to choose as the edges between nodes, and the choice of edges can drastically affect the performance of downstream semi-supervised learning systems. However, despite the importance of graph design, most of the literature assumes that the graph is static.

In this work, we present Grale, a scalable method we have developed to address the problem of graph design for graphs with billions of nodes. Grale operates by fusing together different measures of (potentially weak) similarity to create a graph which exhibits high task-specific homophily between its nodes. Grale is designed for running on large datasets. We have deployed Grale in more than 20 different industrial settings at Google, including datasets which have *tens of billions* of nodes, and *hundreds of trillions* of potential edges to score. By employing locality sensitive hashing techniques, we greatly reduce the number of pairs that need to be scored, allowing us to learn a task specific model and build the associated nearest neighbor graph for such datasets in hours, rather than the days or even weeks that might be required otherwise.

We illustrate this through a case study where we examine the application of Grale to an abuse classification problem on YouTube with hundreds of million of items. In this application, we find that Grale detects a large number of malicious actors on top of hard-coded rules and content classifiers, increasing the total recall by 89% over those approaches alone.

ACM Reference Format:
Jonathan Halcrow†, Alexandru Moşoi‡, Sam Ruth†, Bryan Perozzi† . 2020. Grale: Designing Networks for Graph Learning. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394486.3403302



(a) Observed Similarity (b) Ideal Similarity

Figure 1: The Graph Design Problem. On the left is an example of a typical product graph with several different kinds of possible similarity (edge types) connecting nodes of two types (green,red). While informative, this graph can be challenging for semi-supervised learning methods – even though the products share some facets of similarity, they are not informative for the final task. On the right, is the ideal graph similarity which would maximize label classification performance on this dataset. In this work, we present Grale, a scalable method for extracting graphs from very large datasets.

1 INTRODUCTION

As the size and scope of unlabeled data has grown, there has been a corresponding surge of interest in graph-based methods for Semi-Supervised Learning (SSL) (e.g.[3, 33, 30, 5]) which make use of both the labeled data items and also of the structure present across the whole dataset. An essential component of such SSL algorithms is the *manifold assumption* —the assumption that the labels vary smoothly over the underlying graph structure (i.e. that the graph possesses homophily). As such, the datasets that are typically used to evaluate these methods are small datasets that exhibit this property (e.g. citation networks). However, there is relatively little guidance on how to apply these methods when there is not an obvious source of similarity that's useful for the problem [29].

Real-world applications tend to be multi-modal, and so this lack of guidance is especially problematic. Rather than there being just one simple feature space to choose a similarity on, we have a wealth of different modes to compare similarities in (each of which may be suboptimal in some way [10]). For example, video data comes with visual signals, audio signals, text signals, and other kinds of metadata - each with their own ways of measuring similarity. Through work on many applications at Google we have found that the right answer is to choose a combination of these. But how should we choose this combination?

In this paper we present Grale, a method we have developed to design graphs for SSL. Our work on Grale was motivated by practical problems faced while applying SSL to rich heterogeneous

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08.
https://doi.org/10.1145/3394486.3403302

datasets in an industrial setting. Grale is capable of learning similarity models and building graph for datasets with *billions* of nodes in a matter of hours. It is widely used inside Google, with more than 20 deployments, where it acts as a critical component of semi-supervised learning systems. At its core, Grale provides the answer to a very straightforward question – given a semi-supervised learning task – "do we want this edge in the graph?".

We illustrate the benefits of Grale with a case study of a real anti-abuse detection system at YouTube, a large video sharing site. Fighting abuse on the web can be a challenging problem area because it falls into an area where features that one might have about abusive actors are quite sparse compared to the number of labeled examples available at training time. Abuse on the web is also often done at scale, inevitably leaving some trace community structure that can be exploited to fight it. Standard supervised approaches often fail in this type of setting since there is not enough labeled examples to adequately fit a model with capacity large enough to capture the complexity of the problem.

Specifically, our contributions are the following:

(1) An algorithm and model for learning a task specific graph for semi-supervised applications.
(2) An efficient algorithm for building a nearest neighbor graph with such a model.
(3) A demonstration of the effectiveness of this approach for fighting abuse on YouTube, with hundreds of millions of items.

2 PRELIMINARIES

2.1 Notation

We consider a general multiclass learning setting where we have a partially labeled set of points $\mathcal{X} = \{x_1, x_2, \ldots, x_V\}$ where the first L points have class labels $\mathrm{Y} = \{\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_L\}$, each y_k being a one-hot vector of dimension C, indicating which of the C classes the associated point belongs to. Further, we assume each point $x_i \in \mathcal{X}$ has a multi-modal representation over d modalities, $x_i = \{x_{i,1}, x_{i,2}, \ldots x_{i,d}\}$. Each sub-representation $x_{i,d}$ has its own natural distance measure κ_d.

2.2 Loss functions

Ordinarily one might try to fit a model to make predictions of $y_k = \hat{y}(\mathrm{x_k})$ by selecting a family of functions and finding a member of it which minimizes the cross-entropy loss between y and $\hat{y}$:

$$\mathcal{L} = -\sum_{i \in \mathcal{Y}} \sum_{c \in C} y_{i,c} \log \hat{y}_{i,c}, \tag{1}$$

where $y_{i,c}$ and $\hat{y}_{i,c}$ indicate the ground truth and prediction scores for point i with respect to class c.

Given some graph $G = (V, E)$ with edge weights w_{ij} on our data, SSL graph algorithms [25, 33, 19] nearly universally seek to minimize a Potts model type loss function (either explicitly or implicitly) similar to:

$$\mathcal{L} = \sum_{i,j \in E} w_{i,j} \sum_{c \in C} |\hat{y}_{i,c} - \hat{y}_{j,c}| + \sum_{i \in V, c \in C} |\hat{y}_{i,c} - y_{i,c}|. \tag{2}$$

A classic way to minimize this type of loss is to iteratively apply a label update rule of the form:

$$\hat{y}_{i,c}^{(n+1)} = \alpha y_{i,c} + \beta \frac{\sum_{j \in \mathcal{N}_i} w_{i,j} \hat{y}_{j,c}^{(n)}}{\sum_{j \in \mathcal{N}_i} w_{i,j}}, \tag{3}$$

where $\mathcal{N}_i$ is the neighborhood of point i, $\hat{y}_{i,c}^{(n)}$ is the predicted score for point i with respect to class c after n iterations, and α and β are hyperparameters. Here, we consider a greedy solution which seeks to minimize Eq. (1) using a single iteration of label updates.

2.3 The Graph Design Problem

In Equations (2) and (3) it is assumed the edge weights $w_{i,j}$ are given to us. However in the multi-modal setting we describe here it is not the case that there is a single obvious choice. This is a critical decision when building such a system, which can strongly impact the overall accuracy [10]. In many cases one might heuristically choose some similarity measure, performing some hyperparameter optimization routine to select the number of neighbors to compare to or some ϵ similarity threshold. When one includes ways of selecting or combining various similarity measures, the parameter space can become too large to feasibly handle with a simple grid search - especially when the dataset being operated on becomes large.

Instead we tackle this problem head on, framing the problem as one of graph design rather than graph selection. The Graph Design problem is as follows:

Given:

- A multi-modal feature space $\mathcal{X}$ (as in Subsection 2.1)
- A partial labeling on this feature space $\mathcal{Y}$
- A learning algorithm $\mathcal{A}$ which is a function of some graph G having vertex set equal to the elements of $\mathcal{X}$

Find: An edge weighting function $w(x_i, x_j)$ that allows us to construct a graph G which optimizes the performance of $\mathcal{A}$.

3 GRALE

Grale is built to efficiently solve this type of problem - designing task specific graphs by reducing the original task to training a classifier on pairs of points. We assume the existence of some oracle $y(x_i, x_j)$ which gives a binary valued ground truth, defined for some subset of unordered pairs (x_i, x_j). The oracle is chosen to produce a graph amenable to the primary task, $\mathcal{A}$; in the multiclass learning setting described above, the relevant oracle would return true if and only if both points are labeled and share the same class.

3.1 A Pairwise Loss for Graph Design

We start with the nearest neighbor update rule from Eq. (3) and let the weights be the log of some function of our 'natural' metrics in each mode: $w_{ij} := \log G(x_i, x_j)$, for some $G : \mathrm{R}^d x \mathrm{R}^d \to \mathrm{R}$, which only depends on the d modality distances:

$$G(x_i, x_j) = f(\kappa_1(x_i, x_j), \kappa_2(x_i, x_j), \ldots, \kappa_d(x_i, x_j)). \tag{4}$$

Next, we choose a relaxed form of Eq. (3), where we drop the normalization factor and instead impose a constraint $w_{ij} < 0$

$$\hat{y}_{i,c} = \exp \sum_{j \in \mathcal{N}_{i,c}} w_{ij} \tag{5}$$

with $\mathcal{N}_{i,c}$ being the set of neighbors of node i with $y_{j,c} = 1$.

Removing the normalization factor greatly simplifies the optimization problem. Applied to G, our new constraint is transformed to requiring that $0 < G < 1$. This can be achieved simply by choosing G to belong to some bounded family of functions (such as applying a sigmoid transform to the output of a neural network). Note also that since each κ is a metric and thus symmetric in its arguments this also gives us that $w_{ij} = w_{ji}$, ensuring that a nearest neighbor graph we build with this edge weighting function will be undirected. Putting this back into the full multi-class cross-entropy loss on the node labels Eq. (1), we recover the loss of a somewhat simpler problem to solve: the binary classification problem of deciding whether two nodes are both members of the same class.

$$\mathcal{L} = -\sum_{c\in C}\sum_{i\in\mathcal{X}}\sum_{j\in\mathcal{X}} y_{i,c}, y_{j,c}\log G(x_i, x_j). \tag{6}$$

3.2 Locality Sensitive Hashing

A key requirement for Grale is that it must scale to datasets containing billions of nodes, making an all-pairs search infeasible. Instead we rely on approximate similarity search using locality sensitive hashing (LSH). Generally speaking LSH works by selecting a family of hash functions $\mathcal{H}$ with the property that two points which hash to the same value are likely to be 'close'. Then one compares all such points which share a hash value. In our case, however, the notion of similarity is the output of a model. So we don't have an obvious choice of LSH function. Here we analyze two different ways to construct such a LSH function for our input distances: AND-amplification and OR-amplification.

Recall that above we assume $G(\mathrm{x}_i, \mathrm{x}_j)$ is purely a function of distances between x_i and x_j in D subspaces. Let us define a metric μ on $\mathcal{X}\times\mathcal{X}$ as a sum over our individual metrics κ_d:

$$\mu\big((x_1, x_2), (x_3, x_4)\big) = \sum_d \kappa_d(x_1, x_3) + \kappa_d(x_2, x_4). \tag{7}$$

Let us also assume that our similarity model G is Lipschitz continuous on $\mathcal{X}\times\mathcal{X}$, namely that there exists some $K \in \mathrm{R}$ such that:

$$|G(x_i, x_j) - G(x_k, x_l)| \le K\mu\big((x_i, x_j), (x_k, x_l)\big). \tag{8}$$

This implies that

$$|G(x_i, x_i) - G(x_i, x_j)| \le K\sum_d \kappa_d(x_i, x_j). \tag{9}$$

Next, let $\Delta(x_i, x_j) = 1 - G(x_i, x_j)$. We note that $\Delta(x_i, x_j)$ is not precisely a metric, but it is bounded to the unit interval. Applying this to Eq. (9) yields:

$$|\Delta(x_i, x_i) - \Delta(x_i, x_j)| \le K\sum_d \kappa_d(x_i, x_j). \tag{10}$$

In practice $\Delta(x_i, x_i)$ is quite small (a point always has the same label as itself!), but a learned model may not evaluate to exactly 0. Let $\epsilon = \sup_{x\in\mathcal{X}}\Delta(x, x)$, either $\Delta(x_i, x_j) < \epsilon$ or we have

$$\Delta(x_i, x_j) - \epsilon \le \Delta(x_i, x_j) - \Delta(x_i, x_i) \le K\sum_d \kappa_d(x_i, x_j). \tag{11}$$

In either case,

$$\Delta(x_i, x_j) \le K\sum_d \kappa_d(x_i, x_j) + \epsilon. \tag{12}$$

Next let $\mathcal{H}_n$ be an (r, cr, p, q) sensitive family of locality sensitive hash functions for κ_n. That is, two points x_i and x_j with $\kappa_n(x_i, x_j) \le r$ have probability at least p of sharing the same hash value for some hash function h randomly chosen from $\mathcal{H}_n$. Further two points x_i and x_j with $\kappa_n(x_i, x_j) \ge cr$ have probability of at most q of sharing the same hash value for a similarly chosen h.

We construct a new 'AND' family $\mathcal{H}_\otimes$ of hash functions by concatenating hash functions from each of the $\mathcal{H}_n$, with each being $(r, c_n r, p_n, q_n)$ sensitive for the associated space. Then $\mathcal{H}_\otimes$ is $(r_\otimes, cr_\otimes, p_\otimes, q_\otimes)$-sensitive for G, where

$$\begin{aligned} r_\otimes &= KrD + D\epsilon \\ p_\otimes &= \prod_d p_d \\ q_\otimes &= \prod_d q_d. \end{aligned} \tag{13}$$

Alternatively we may also employ an 'OR'-style construction of an LSH function, taking $\mathcal{H}_\oplus$ as the union of the $\mathcal{H}_n$. If we assume that there exists some correlation between the subspaces of $\mathcal{X}$ such that two points having $\kappa_m(x_i, x_j) \le r$ also have $\kappa_n(x_i, x_j) \le r$ with probability p_{mn} (and likewise for distance $\ge cr$ with probability q_{mn}) then $\mathcal{H}_m$ acts as a locality sensitive hash function for $\mathcal{H}_n$. Under this assumption $\mathcal{H}_\oplus$ is $(r_\oplus, cr_\oplus, p_\oplus, q_\oplus)$-sensitive for G, where

$$\begin{aligned} r_\oplus &= KrD + D\epsilon \\ p_\oplus &= \min_d \left(1 - (1 - p_d)\prod_{l\neq d}(1 - p_{dl})\right) \\ q_\oplus &= \max_d \left(1 - (1 - q_d)\prod_{l\neq d}(1 - q_{dl})\right). \end{aligned} \tag{14}$$

In practice, we employ a mixture of the AND and OR forms of locality sensitive hash functions when applying Grale, tuning choices of functions and hyper-parameters by grid search (see Table 2 for an evaluation of the performance on a real world dataset).

3.3 Graph building algorithm

Our graph building algorithm works by employing the LSH function constructed for our similarity measure to produce candidate pairs, then scoring the candidate pairs with the model. More explicitly, we start by sampling S hash functions from our chosen family $\mathcal{H}_s$ (constructed from the set of LSH families for our features). Then for each point $p_i \in X$ we compute a set of hashes, bucketing by hash value. Finally, we find it practical to cap the size of each bucket (typically this is chosen to be 100), randomly subdividing any bucket which is larger. Without this step in the worse case we could still be computing all $O(N^2)$ comparisons. An additional practical step we take in some cases is to simply drop buckets which are too large, since this is usually indicative of hashing on a 'noise' feature and this buckets tend not to contain many desirable pairs, and often lead to false positives from the model. For each pair in our subdivided buckets we compare all pairs with our scoring function G (learned using the training procedure described in the next section). Finally we output the top k values (potentially subject to some ϵ threshold on the scores).

Function NNSketching($\mathcal{X}$):
  Input : A set of points $\mathcal{X}$
  foreach $p \in \mathcal{X}$ **do**
    foreach $h \in \mathrm{LSH}(p)$ **do**
      Append h, p to sketches;
    end
  end
  Sort sketches by hash value ;
  Collect sketches into bucket with same hash → Buckets;
  Subdivide buckets in Buckets of size larger than K;
  return Buckets;

Function BuildGraph($\mathcal{X}$, $\hat{y}$, ϵ):
  Input : A set of points $\mathcal{X}$, similarity model $\hat{y}$, and minimum edge weight ϵ
  Output: A nearest neighbor graph G
  Buckets = NNSketching($\mathcal{X}$)
  foreach bucket ∈ Buckets **do**
    foreach *pair* $p_i, p_j \in$ bucket **do**
      $w_{ij} = \hat{y}(p_i, p_j)$;
      if $w_{ij} > \epsilon$ **then**
        Emit edge (p_i, p_j, w_{ij}) ;
      end
    end
  end
  return G = ∪ *edges*

Algorithm 2: Graph Building Algorithm

Training algorithm for Grale

During the learning phase, we follow the same algorithm described in the previous section, but instead of applying the model we instead consult the user supplied oracle $y(p_i, p_j)$. Of course, the oracle will not be able to supply a value for every pair - if so there would be no need for a model! In the cases where the oracle does have some ground truth we save the pairwise feature vector δ_{ij} and the value returned by the oracle, to generate training and hold-out sets Δ_1, Δ_2. This algorithm is detailed in the Appendix.

We then train the model on the saved pairs, holding out some of the pairs for model evaluation. Note: it is important to perform the holdout by splitting the set of points rather than pairs so that data from the same point does not appear in both the training set and holdout set to avoid over-fitting. Our model is optimized for log-loss, namely we seek to minimize the following quantity, with $\hat{y}_{ij}$ being the confidence score of our model (described in section 3.5):

$$L = -\frac{1}{|\Delta|} \sum_{i,j \in \Delta} y_{i,j} \log \hat{y_{ij}} + (1 - y_{ij}) \log (1 - \hat{y}_{ij}). \qquad (15)$$

3.4 Jointly learned node embeddings

So far we have focused on the setting where we only use per-mode distance scores as features, but this approach can be extended to also jointly learn embeddings of the nodes themselves. In the extension we divide the model into two components: the node embedding component and the similarity learning component. The distances input to the similarity learning component are augmented with per-dimension distances from the embeddings. By training the embedding model with the similarity model, we can backpropagate the similarity loss to the embedding layers obtaining an embedding tailored for the task.

Augmenting the model in this way allows us to employ 'two-tower' style networks[4] to learn similarity for features where that technique works well (for example image data [14]), but use pure distances for sparser features where the embedding approach does not work as well. In some cases we may even rely entirely on distances derived from the embeddings (as is the case for some of the experiments in Section 4). We note that the LSH family used above can still be applied in this setting assuming that the embedding function is Lipschitz (following a similar argument as in Section 3.2).

3.5 Model Structure

The structure of the model used by Grale can in general vary from application to application. In practice we have used Grale with both neural networks and gradient boosted decision tree models [24, 23]. Here we will describe the neural network approach in more detail.

The architecture of the neural net that we use is given in Figure 2. The inputs to the neural net are split into two types: pairwise features which are functions of the features of both nodes (δ_{ij} above), and singleton features - features of each node by itself. Examples of pairwise features might include the Jaccard similarity of tokens associated with two nodes. We use a standard 'two-tower' architecture for the individual node features, sharing weights between each tower, to build an embedding of each node. These embeddings are multiplied pointwise and concatenated with the vector of distance features $\kappa_1(x_i, x_j), \ldots, \kappa_d(x_i, x_j)$ to generate a combined pairwise set of features. We choose a pairwise multiplication here because it guarantees that the model is symmetric in its arguments (i.e. $\hat{y}_{ij} = \hat{y}_{ji}$). Note that since the rest of the model takes distances as inputs, it is symmetric by construction.

Note that here we have extended G to be a function of not just the original distances but also of the input points themselves. However, if we consider the embeddings learned as an augmentation of the data itself, then we are back to the case of a pure function of distances, where we have also included coordinate-wise distances in the embedded space. We can also argue that the locality sensitive hash functions for distances in the un-embedded versions of these spaces cover as LSH functions in the embedded space due to the correlation between the two.

3.6 Complexity

Without the LSH process, the complexity of graph building on a set of N points would be $O(N^2)$ (since we compare each point to every other). However instead, employing LSH we compute S hashes per point. Grouping points by hash value is done by sorting, add a factor of $O(SN \log N)$ To cap the amount of work done per bucket, we further subdivide these buckets into smaller subbuckets of maximum size B (100 is a typical value). Only then do we compare all points within the same subbucket, giving us a complexity of $O(SNB)$ for this step. Putting this together and doing a

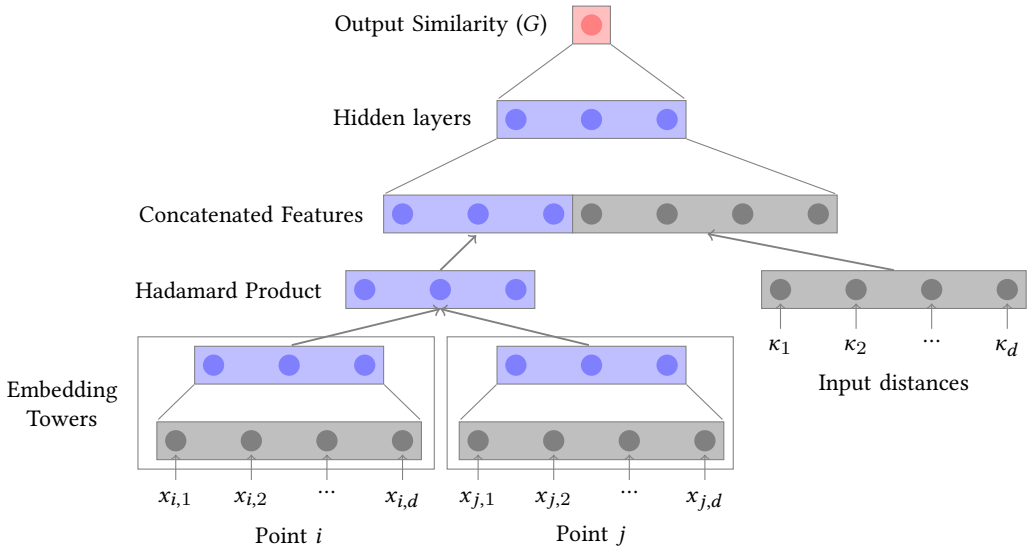


Figure 2: The Grale Neural Network model. The gray nodes are inputs, the blue are hidden layers, and red is the output. The network architecture combines a standard two-tower model with natural distances in the input feature spaces. Weights are shared between the two towers. The Hadamard product (pointwise multiplication) of the two towers is used to give us a pairwise embedding. We treat this as an additional set of distance features to augment the input distance features $\kappa_1(x_i, x_j), \ldots, \kappa_d(x_i, x_j)$. This combined set acts as an input to the second part of the model which computes a single similarity score.

fixed amount of work per comparison, giving us a final complexity of $O(SN(B + \log N))$ work to score all of the edges.

4 EVALUATION ON PUBLIC DATASETS

4.1 Experimental Design

There are many papers describing improved techniques for semi-supervised learning. We focus on [29] as a comparison point since it also describes an approach to learning a task specific similarity measure, called PG-Learn. PG-Learn learns a set of per dimension bandwidths for a generalized version of the RBF kernel

$$K(x_i, x_j) = \exp\left(-\sum_m \frac{(x_{im} - x_{jm})^2}{\sigma_m}\right). \tag{16}$$

Rather than directly optimizing this as we do in this paper, PG-Learn sets out to optimize it as a similarity kernel for the Local and Global Consistency (LGC) algorithm by Zhou et al [33].

We compare on 2 of the same datasets used by [29]. USPS[1] is a handwritten digit set, scanned from envelopes by the U.S. postal service and represented as numeric pixel values. MNIST[2] is another popular handwritten digit dataset, where the images have been size-normalized and centered. Both tasks use a small sample of the labels to learn from, to more realistically simulate the SSL setting. We list the size, dimension, and number of classes for these datasets in the appendix.

To build a similarity model with Grale, we select an oracle function that, for a pair of training nodes, the edge weight between them is 1 if they are in the same class and 0 otherwise. Once trained, we use this model to compute an edge weight for every pair of vertices. Since the number of nodes in these datasets is small compared to the number of nodes in a typical Grale application, we computed the edge weights for all possible pairs of nodes; we did not use locality sensitive hashing.

We then use the graph as input to Equation 2 and label nodes by computing a score for each class as weighted nearest neighbors score, weighting by the log of the similarity scores. We choose the class with the highest score as the predicted label. That is,

$$\max_{c \in C} \exp\left(\sum_{j \in N_c(i)} \log G_{ij}\right), \tag{17}$$

where $N_c(i)$ is the set of nodes in the neighborhood of node i that appear in our training set with class c, and G_{ij} is our model's similarity score for the pair. Many of these other approaches also build a k-NN graph, with distance defined by a kernel:

In Table 1, we list the results of using Grale for these classification tasks alongside the results from [29], which include PG-Learn, alongside some other approaches.

- Grid search (GS): k-NN graph with RBF kernel where k and bandwidth σ are chosen via grid search,
- $Rand_d$ search (RS): k-NN with the generalized RBF kernel where k and different bandwidths are randomly chosen,
- MinEnt: Minimum Entropy based tuning of the bandwidths as proposed by Zhu et al [34],
- AEW: Adaptive Edge Weighting by Karasuyama et al. that estimates the bandwidths through local linear reconstruction [12].

For every result in this table, 10% of the data was used for training. For all results except Grale, there was 15 minutes of automatic hyperparameter tuning. For Grale, we manually tested different edge weight models for each dataset. For MNIST we used a two-tower DNN with two fully connected hidden layers. For USPS we used CNN towers.

[1] http://www.cs.huji.ac.il/~shais/datasets/ClassificationDatasets.html
[2] http://yann.lecun.com/exdb/mnist/

Dataset	**Grale**	**PGLrn**	**MinEnt**	**AEW**	**Grid**	**Rand$_d$**
USPS	0.892	0.906	0.908	0.895	0.873	0.816
MNIST	0.927	0.824	0.816	0.782	0.755	0.732

Table 1: Test Accuracy of Various Methods

In Table 1 we see that while Grale does not always give the best performance it produces results that are fairly similar to several other methods. In this case, the number of labels is so small that we found it quite easy to overfit in similarity model training. It is likely that in this case, all methods are learning roughly the same thing. This is especially true given how competitive Grid is, where the similarity has a single free parameter. On the other hand, MNIST provides quite a bit more to work with, so Grale is able to learn quite a bit and strongly outperforms the other methods.

5 CASE STUDY: YOUTUBE ABUSE DETECTION

The comparisons in Section 4 give some sense for how Grale performs; however, the datasets considered are much smaller than what Grale was designed to work with. They are also single mode problems, where the features belong to a single dense feature space. We provide a case study here on how Grale was applied to abuse on YouTube, which is much more on the scale of problem that Grale was built to solve.

YouTube is a video sharing platform with hundreds of millions of active users [31]. Its popularity makes it a common target [11] for organizations seeking to post to spam [15, 28, 17]. Between April 2019 and June 2019, YouTube removed 4M channels for violating its Community Guidelines [26].

5.1 Experimental Setup

We trained Grale with ground truth given by Equation 18, treating items as either belong to the class of abusive items (removed for Community Guidelines violations) or the class of safe (active, non-abusive) items. We also ignore the connections between safe (non-abusive) pairs during training since these are not important for propagating abusive labels on the graph.

$$y_{ij} = \begin{cases} 1 & \text{if items } i \text{ and } j \text{ are abusive} \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

Experiments were done using both the neural network and tree-based formulations of Grale, but we found better performance with the tree version.

5.1.1 LSH Evaluation. In our many applications of Grale, we have observed that the choice of locality sensitive hash functions is critical to the overall performance of the system. There is some trade-off between p, the floor on the likelihood of producing close pairs, and q, the ceiling of the likelihood of producing far pairs. Having too large q results in graph building times that are too long to be useful, resulting in too much delay before items could be taken down. Alternatively, if p is too small we miss too many connections, hurting the overall recall of the entire system.

Here we consider two different thresholds in evaluating our LSH scheme. The first (r in Section 3.2) is a threshold chosen such that a connection at that strength is strongly indicative of an abusive item. The second ('cr') is a more moderate threshold, chosen such that the connections are still interesting and useful when combined with some other information in determining if an item is abusive. The final numbers we arrived at are given in Table 2. In order to find the same number of high weight edges using random sampling, we would need to evaluate $10.5x$ more edges than we currently do (note this is assuming sampling without replacement, whereas LSH finds duplicates).

	% strong ties (p)	% weak ties (q)
baseline (random pairs)	0.0653%	77.4%
tuned LSH	52.3%	22.7%

Table 2: A comparison of the LSH function used for YouTube and a naive baseline. The parameters p and q are the same as defined in Section 3.2: % of pairs returned by LSH with distance less than r where r is chosen to be a high precision decision threshold of the model and % of pairs returned by LSH with distance further than a moderate precision decision threshold, respectively.

	degree 0	degree >0	high-precision degree >0
safe (100%)	87.80%	12.20%	2.32%
abusive (100%)	51.87%	48.13%	36.96%

Table 3: Percentage of items safe and abusive that have a node in the graph, considering only high-precision edges. Abusive nodes are 6-16x more likely to be connected in the graph.

5.2 Graph Analysis

After finding a suitable hashing scheme and model for the dataset, we can materialize the graph. Here we provide a quantitative analysis of the global properties of the graph. In Section 5.4, we visualize the graph for additional insights.

We set a threshold on edge weights to guarantee a very high level of precision. This means that many nodes end up without any neighbors in the graph. Table 3 shows that after pruning 0-degree nodes (i.e. those without a very high confidence neighbor) the graph covers 36.96% of the abusive nodes, but only 2.31% of the safe nodes.

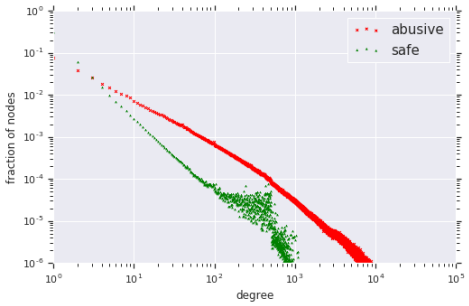
Figure 3a shows the distribution of node degrees, separated into safe and abusive items. Abusive nodes generally have more neighbors than safe nodes. However simply having related items doesn't make an item abusive. Safe nodes may have other safe neighbors for legitimate reasons. So it is important to take into account more than just the blind structure of a node's neighborhood.

Similarly, Figure 3b shows the degree of safe and abusive items, but only when the neighbors are restricted to abusive nodes. Safe nodes can have abusive neighbors if the edge evaluation model is imprecise or the labels are noisy. However there is a clear drop-off in the frequency of high-degree safe nodes from Figure 3a.

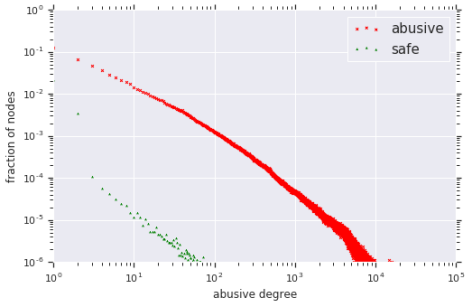
5.3 Classification Results

Next, we performed a graph classification experiment using Grale. To show the utility of graphs designed by our method, we built and evaluated a simple single nearest neighbor classifier.[3] We selected

[3]We note that in practice, a more complex system is employed, but the details of this are beyond the scope of this paper.



(a) Node degree distribution in Grale YouTube.



(b) Abusive-degree distribution in Grale YouTube.

Figure 3: Degree distributions in the learned YouTube graph. In 3a, the total degree distribution for nodes, broken down by abusive status. In 3b, the number of abusive neighbors for safe (green) and abusive (red) nodes. We see that safe nodes have far fewer abusive neighbors on average.

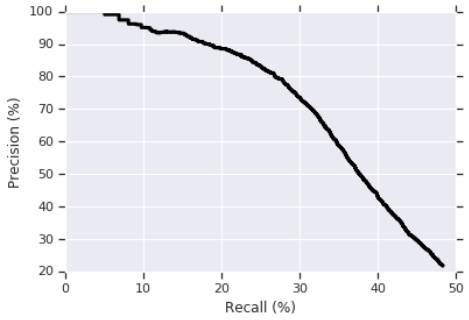


Figure 4: Precision-recall curve of single nearest-neighbor classification on our graph. We selected 25% of the abusive items as node seeds, and propagated the label to neighboring nodes.

25% of the abusive nodes as seeds, and for every other node in the graph we assigned a score equal to the maximum weight of the edges connecting the node to a seed. Also, to simulate more real-world conditions we split our training and evaluation sets by time, training the model on older data but evaluating on newer data. This gives us a more accurate assessment of how the system would function after deployment, since we will always be operating on data generated after the model was trained. The precision-recall curve of this classifier can be observed in Figure 4.

Algorithm	% of items
Content Classifiers	47.7%
Heuristics	5.3%
Total (Heuristics + Content Classifiers)	53%
Grale+Label Propagation	47.0% **(+89%)**

Table 4: Abuse identified from different techniques on a subset of YouTube abusive items. Adding Grale to the existing system increased recall by 89%.

Algorithm	New items	Old items
Grale+Label Propagation	25.8%	74.2%
Content Classifiers	71.6%	28.4%
Heuristics	33.7%	66.3%

Table 5: Breakdown by item age at the time of classification as abusive across various methods. 'Grale + Label Propagation is able to identify additional items initially missed by content classifiers.

We compared Grale+Label Propagation, content classifiers, and some heuristics against a subset of YouTube abusive items. Content classifiers were trained using the same training data to predict abusive/safe.

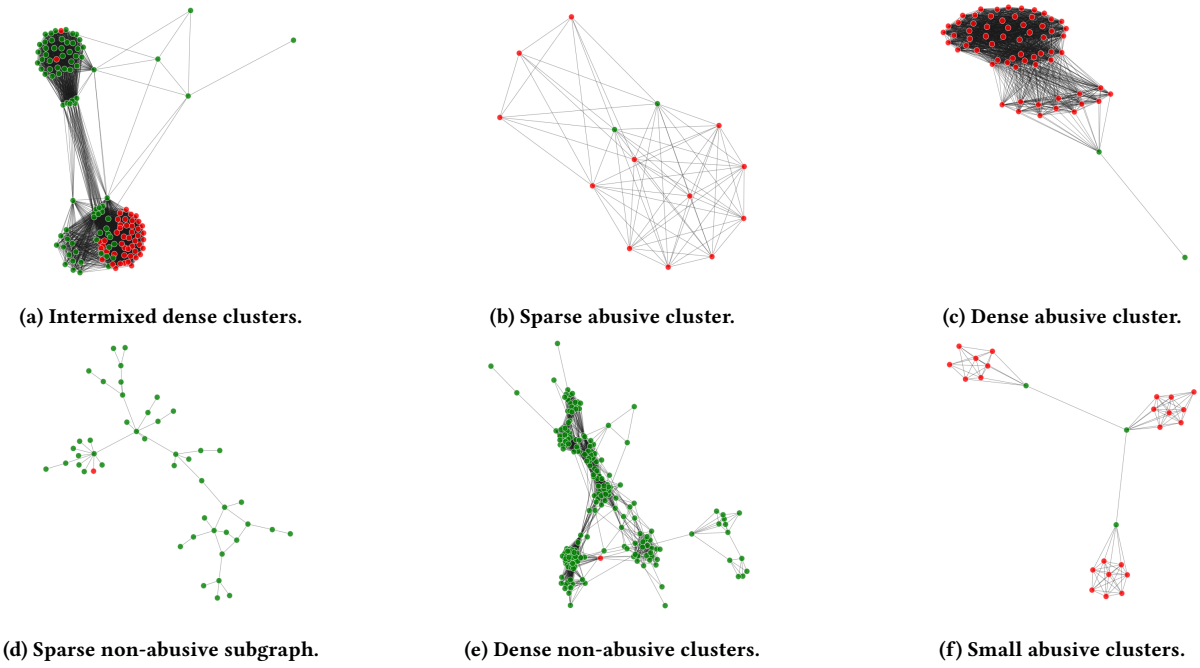
The results in Table 4 show that Grale was responsible for half of the recall. While content classifiers were given the first chance to detect an item (for technical reasons), Figure 5 shows that content classifiers miss a significant part of recall when faced with new trends which the classifiers have not been trained on. With Grale, the labels can be propagated to older items as well without requiring new model retraining.

5.4 Graph Visualization

Finally, to give a sense of how well the scope and variety of structure present in graphs designed by Grale, here we provide two kinds of visualizations of the local and global structure of the YouTube graph.

5.4.1 Local structure. Figure 5 shows visualization of several subgraphs extracted from the YouTube graph. The resulting visualizations illustrate the variety of different structure present in the data. For instance in Figure 5a, we see that clear community structure is visible – there is one tight cluster of abusive nodes (marked in red) and one tight cluster of non-abusive (in green). We note how this figure (along with Fig. 5e) illustrates the earlier point that existing in a dense neighborhood in the graph, while potentially suspicious, is not necessarily an indicator of abuse. Simultaneously, we can see that the resulting graph can contain components with relatively small or large diameter (Fig. 5c and Fig. 5d respectively).

5.4.2 Global structure. In order to visualize the global structure of the YouTube case study, we trained a random walk embedding [20, 1] on the YouTube graph learned through Grale. Figure 6 shows a two-dimensional t-SNE [16] projection of this embedding. We observe that, in addition to the rich local structure highlighted above, the Grale graph contains interesting global structure. In order to understand the global structure of the graph better, we performed a manual inspection of dense regions of the embedding space, and found they corresponded to known abusive behavior.



(a) Intermixed dense clusters. (b) Sparse abusive cluster. (c) Dense abusive cluster.

(d) Sparse non-abusive subgraph. (e) Dense non-abusive clusters. (f) Small abusive clusters.

Figure 5: Different subgraphs extracted from the YouTube case study. Here we show a small sample of the rich patterns Grale is able to capture when applied to real data. Colors correspond to the abuse status of the nodes.

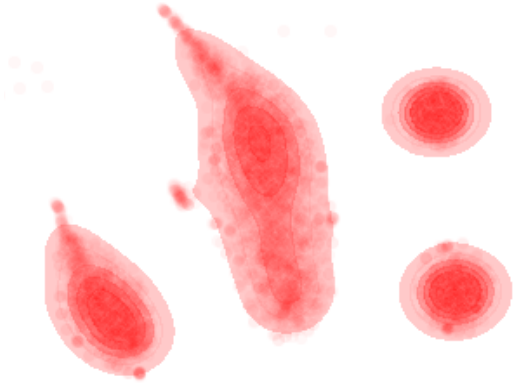


Figure 6: Visualization of global structure. In order to visualize the global structure of the YouTube case study, we learned a random walk embedding [20, 1] of the highest weight edges in the graph. Here we show a t-SNE projection of a sample of these embeddings. Clusters correspond to known abusive behavior.

6 RELATED WORK

We begin by noting that the literature on label inference is vastly larger than that of graph construction. In this section, we briefly discuss the relevant literature in graph construction, which we divide into unsupervised and supervised approaches.

6.1 Unsupervised Graph Construction

Perhaps the most natural form of graph construction is to take nearest neighbors in the underlying metric space. The two straightforward approaches to generate the neighborhood for each data item v either connect it to the k-nearest data items, or to all neighbors within an ϵ-ball. While such *proximity graphs* have some desirable properties, such methods have difficulty adapting to multimodal data.

For large datasets, the naive approach of comparing each point to every other point to find nearest neighbors becomes intractably slow. Locality sensitive hashing is a classic technique for doing approximate nearest neighbor search in metric spaces, speeding up the process significantly. The simplest version of this algorithm would be to compare all pairs which share a hash value. However this still has worst-case complexity of $O(N^2)$ (N being the number of points in question). Zhang et al [32] propose a modification of this algorithm which reduces the complexity to $O(N \log N)$ by sorting points by hash value and executing a brute force search within fixed sized windows in this ordered list.

6.2 Supervised Graph Construction

A second branch of the literature seeks to use training labels to learn similarity scores between a set of nodes [2]. Motivated by tasks like item recommendation [8], these methods aim to model the relationship between data items [7, 6, 9], connecting new nodes or even removing incorrect edges from an existing graph [22]. While Grale could be thought of as a link prediction system, our motivations are substantially different. Instead of finding new connections between data items, we seek to accurately characterize the many possible interactions into the most useful form of similarity for a given problem domain. Other work focuses on modeling the joint distribution between the data items and their label for better semi-supervised learning [13]. Salimans et al. [27] use a generative adversarial network (GAN) to model this distribution.

These approaches typically have good performance on small-scale SSL tasks, but are not scalable enough for the size or scale of real datasets, which may contain millions of distinct points. Unlike this work, Grale is designed to be scalable and therefore utilizes a conditional distribution between the data and labels.

Perhaps the most relevant related work comes from Wu et al. [29], who propose a kernel for graph construction which reweights the columns of the feature matrix. Unlike this approach, Grale is capable of modeling arbitrary transformations over the feature space, operating on complex multi-modal features (like video and audio), and running on billions of data items. Similarly, work on clustering attributed graphs have explored attributed subspaces [18] as a form of graph similarity, for example, using distance metric learning to learn edge weights [21].

7 CONCLUSION

In this paper we have demonstrated Grale, a system for learning task specific similarity functions and building the associated graphs in multi-modal settings with limited data. Grale scales up to handling a graph with billions of nodes thanks to the use of locality sensitive hashing, greatly reducing the number of pairs that need to be considered.

We have also demonstrated the performance of Grale in two ways. First, on smaller academic datasets, we have shown that it is capable of producing competitive results in settings with limited amounts of labeling. Second, we have demonstrated the capability and effectiveness of Grale on a real YouTube dataset with hundreds of millions of data items.

ACKNOWLEDGEMENTS

We thank Warren Schudy, Vahab Mirrokni, Natalia Ponomareva, Peter Englert, Filipe Miguel Gonçalves de Almeida, and Anton Tsitsulin.

REFERENCES

[1] S. Abu-El-Haija et al. 2017. Learning edge representations via low-rank asymmetric projections. *CIKM*.
[2] M. Al Hasan et al. 2006. Link prediction using supervised learning. *SDM Workshops*.
[3] A. Blum et al. 2001. Learning from labeled and unlabeled data using graph mincuts. *ICML*.
[4] J. Bromley et al. 1994. Signature verification using a" siamese" time delay neural network. *NIPS*.
[5] I. Chami et al. 2020. Machine learning on graphs: a model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675*.
[6] H. Chen et al. 2018. A tutorial on network embeddings. *arXiv preprint arXiv:1808.02590*.
[7] H. Chen et al. 2018. Enhanced network embeddings via exploiting edge labels. *CIKM*.
[8] H. Chen et al. 2005. Link prediction approach to collaborative filtering. *JCDL*.
[9] P. Cui et al. 2018. A survey on network embedding. *TKDE*.
[10] C. A. R. de Sousa et al. 2013. Influence of graph construction on semi-supervised learning. *ECML/PKDD*.
[11] C. Kanich et al. 2011. Show me the money: characterizing spam-advertised revenue. *SEC*.
[12] M. Karasuyama et al. 2017. Adaptive edge weighting for graph-based learning algorithms. *Mach. Learn.*
[13] D. P. Kingma et al. 2014. Semi-supervised learning with deep generative models. *NIPS*.
[14] G. Koch. 2015. Siamese neural networks for one-shot image recognition. *ICML Workshops*.
[15] K. Levchenko et al. 2011. Click trajectories: end-to-end analysis of the spam value chain. *S&P*.
[16] L. v. d. Maaten et al. 2008. Visualizing data using t-sne. *JMLR*.
[17] D. McCoy et al. 2012. Pharmaleaks: understanding the business of online pharmaceutical affiliate programs. *SEC*.
[18] E. Müller et al. 2009. Evaluating clustering in subspace projections of high dimensional data. *VLDB*.
[19] A. Murua et al. 2008. On potts model clustering, kernel k-means and density estimation. *Journal of Computational and Graphical Statistics*.
[20] B. Perozzi et al. 2014. Deepwalk: online learning of social representations. *KDD*.
[21] B. Perozzi et al. 2014. Focused clustering and outlier detection in large attributed graphs. *KDD*.
[22] B. Perozzi et al. 2016. When recommendation goes wrong: anomalous link discovery in recommendation networks. *KDD*.
[23] N. Ponomareva et al. 2017. Compact multi-class boosted trees. *Big Data*.
[24] N. Ponomareva et al. 2017. Tf boosted trees: a scalable tensorflow based framework for gradient boosting. *ECML/PKDD*. Y. Altun et al., editors.
[25] S. Ravi et al. 2016. Large scale distributed semi-supervised learning using streaming approximation. *Artificial Intelligence and Statistics*, 519–528.
[26] G. T. Report. [n. d.] https://transparencyreport.google.com/youtube-policy/removals. ().
[27] T. Salimans et al. 2016. Improved techniques for training gans. *NIPS*.
[28] D. Samosseiko. 2009. The partnerka—what is it, and why should you care. *Virus Bulletin Conference*.
[29] X. Wu et al. 2018. A quest for structure: jointly learning the graph structure and semi-supervised classification. *CIKM*.
[30] Z. Yang et al. 2016. Revisiting semi-supervised learning with graph embeddings. *ICML*.
[31] YouTube. [n. d.] https://www.youtube.com/intl/en-GB/about/press/. ().
[32] Y.-M. Zhang et al. 2013. Fast knn graph construction with locality sensitive hashing. *ECML/PKKD*.
[33] D. Zhou et al. 2003. Learning with local and global consistency. *NIPS*.
[34] X. Zhu et al. 2003. Semi-supervised learning using gaussian fields and harmonic functions. *ICML*.

APPENDIX

Training Procedure

Algorithm 3 shows how the sketching function is used in the train/test split.

Input : A set of points $\mathcal{X}$ and oracle function Y
Output: A similarity model on approximating Y on $\mathcal{X}$

Split $\mathcal{X}$ into training and hold-out sets, $\mathcal{X}_{train}, \mathcal{X}_{test}$;
Buckets = $\mathrm{NNSketching}(\mathcal{X})$;
$\mathcal{X}_{train}$ = Buckets $\cap\, \mathcal{X}_{train}$;
$\mathcal{X}_{test}$ = Buckets $\cap\, \mathcal{X}_{test}$;
/* Pairs which the oracle cannot decide are ignored during training */
$\mathcal{Y}_{train} = \mathrm{Y}(\mathcal{X}_{train})$;
$\mathcal{Y}_{test} = \mathrm{Y}(\mathcal{X}_{test})$;
Fit $\hat{y}$ on $(\mathcal{X}_{train}, \mathcal{Y}_{train})$;
Evaluate model on $(\mathcal{X}_{test}, \mathcal{Y}_{test})$;
return $\hat{y}$;

Algorithm 3: Training procedure used for Grale, making using of the NNSketching function from Algorithm 2. Pairs which are not scorable by the oracle are omitted from its output. The model returned by this procedure can then be used with Algorithm 2 to build a nearest neighbor graph.

Dataset details

Some details about the datasets used in Section 4 are described here.

Name	# points	# dimension	# classes	Description
USPS	1000	256	10	Handwritten digits
MNIST	70000	784	10	Handwritten digits

Table 6: Datasets