GOGGLES: Automatic Training Data Generation with Affinity Coding

Nilaksh Das
Georgia Institute of Technology
nilakshdas@gatech.edu

Sanya Chaba
Georgia Institute of Technology
sanyachaba@gatech.edu

Sakshi Gandhi
Georgia Institute of Technology
sakshi@gatech.edu

Duen Horng Chau
Georgia Institute of Technology
polo@gatech.edu

Xu Chu
Georgia Institute of Technology
xu.chu@cc.gatech.edu

ABSTRACT

Generating large labeled training data is becoming the biggest bottleneck in building and deploying supervised machine learning models. Recently, data programming has been proposed in the data management community to reduce the human cost in training data generation. Data programming expects users to write a set of labeling functions, each of which is a weak supervision source that labels a subset of data points with better-than-random accuracy. However, the success of data programming heavily depends on the quality (in terms of both accuracy and coverage) of the labeling functions that users still need to design manually.

We propose *affinity coding*, a new paradigm for fully automatic generation of training data. In affinity coding, the similarity between the unlabeled instances and prototypes that are derived from the same unlabeled instances serve as signals (or sources of weak supervision) for determining class membership. We term this implicit similarity as the affinity score. Consequently, we can have as many sources of weak supervision as the number of unlabeled data points, without any human input. We also propose a system called GOGGLES that is an implementation of affinity coding for labeling image datasets. GOGGLES features novel techniques for deriving affinity scores from image datasets based on "semantic prototypes" extracted from convolutional neural nets, as well as an expectation-maximization approach for performing class label inference based on the computed affinity scores.

Our experiments on the Caltech-UCSD Birds-200-2011 dataset and the Animals with Attributes 2 dataset show that labels generated by GOGGLES have an average accuracy of 98.09% and 92.66% for the binary labeling task, and 92.93% and 83.44% for the multi-class labeling task. Compared to the state-of-the-art data programming system Snorkel, GOGGLES exhibits 14.88% average improvement in terms of the quality of labels generated for the binary labeling task. The GOGGLES system is open-sourced at `https://github.com/chu-data-lab/GOGGLES/`.

1. INTRODUCTION

Machine learning (ML) and deep learning are being increasingly used by organizations to gain insights from data

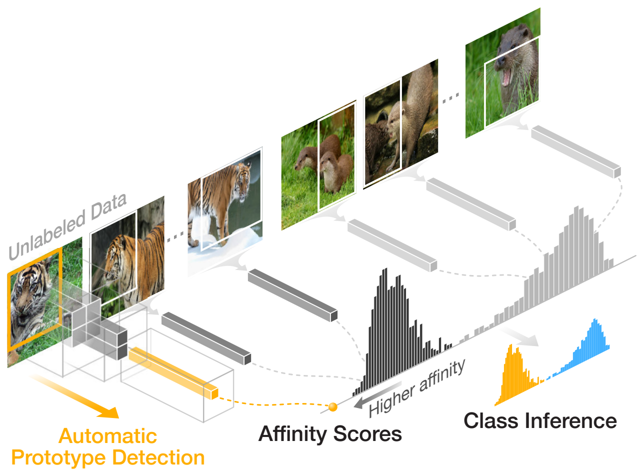

Figure 1: The GOGGLES system. Our approach automatically generates probabilistic labels for unlabeled image datasets without requiring *any* human annotation.

and to solve a diverse set of important problems, ranging from traditional applications such as fraud detection, product recommendation, and customer churn prediction, to more challenging and modern applications such as image recognition, natural language understanding, and even enabling health care and self-driving cars. A fundamental necessity for the success of ML algorithms is the existence of large high-quality labeled training data. For example, the current ConvNet revolution would not be possible without big labeled datasets such as the 1M labeled images from ImageNet [Russakovsky et al., 2015]. Modern deep learning methods often need tens of thousands to millions of training examples to reach peak predictive performance [Sun et al., 2017]. However, for many real-world applications, large hand-labeled training datasets either do not exist, or is extremely expensive to create as manually labeling data usually requires domain experts [Davis et al., 2013].

Existing Work. We are certainly not the first to recognize the need for addressing the challenges arising from the lack of sufficient training data. The ML community has been dealing with it, mostly by modifying how the models are trained, such as by using semi-supervised learning [Zhu, 2005] and transfer learning [Pan and Yang, 2010], which we discuss further in Section 6. Only recently, the *data programming* paradigm [Ratner et al., 2016] and the Snorkel system [Ratner et al., 2017], which is an implementation of the paradigm, were proposed in the data management community that aims at directly addressing the problem by reducing the human effort in generating labeled training data. Snorkel consists of two main steps: (1) users write labeling functions, each of which is a weak supervision source that assigns potentially noisy labels to a subset of data points in the training set. This produces a labeling matrix Λ, where $\Lambda[i, j] \in \{-1, 1, 0\}$ stores the label assigned by source j for x_i (0 denotes source j fails to label x_i). (2) As different labeling functions may provide conflicting signals, the labeling matrix Λ is then de-noised to generate one probabilistic label $\tilde{y}_i$ for every x_i. Unsurprisingly, the quality of the assigned labels depends on the accuracy and coverage of the hand-designed labeling functions, as well as the de-noising algorithm.

Our Proposal and Challenges. Snorkel's main limitation is that users still need to be domain experts in the labeling task, and need to provide many high-quality labeling functions to achieve high labeling accuracy. For example, Snorkel was developed labeling text data, where users can write labeling functions using raw features from text. However, it is difficult for users to write labeling functions for images as the raw features of images (pixels) are intractable for such a task. While Snorkel's primary objective is to reduce the human effort for fast training data generation, we aim for a more ambitious goal of generating training data completely automatically. This is very challenging as we need to completely re-think how we generate the weak supervision sources without any human input. In other words, the unlabeled data must somehow itself serve as the sources of weak supervision to determine the class membership.

Contributions. We make the following contributions that address the above challenges for automatic training data generation.

- **The affinity coding paradigm**. We propose *affinity coding*, a new paradigm for automatic generation of training data. The core premise of affinity coding is that *instances belonging to the same class share certain similarities, which we call affinity, that instances belonging to different classes do not have.* The paradigm thus includes two main components: how to encode the affinity between unlabeled instances that can reflect the class separation, and how to use the computed affinities for class label inference.

  To the best of our knowledge, we are the first to propose a general approach to label training data using only the unlabeled data and without any further inputs, such as a small labeled development set or human annotations.

- **Coding affinity for image datasets.** We propose GOGGLES, a system that implements affinity coding for automatic image labeling. We choose image labeling as our first foray into affinity coding because it is much more difficult for Snorkel to label images as discussed before. GOGGLES must handle the challenges associated with labeling images: (1) noisy raw pixels have high variance even for images in the same class; (2) higher-order signals are unknown in the absence of any labeled data; and (3) a useful signal may reside in different regions of images from the same class.

  To tackle these difficulties, GOGGLES features a novel approach to automatically select the most useful "prototypes" from unlabeled instances. Figure 1 shows one such extracted prototype for the leftmost image. The affinity scores are then calculated between other unlabeled instances and the prototype extracted from an image. It can be seen from Figure 1 that images that are in the same class as the prototype tend to have higher affinity scores than images that are in a different class.

- **Class inference from affinity scores.** Given the affinity scores, GOGGLES proposes techniques to perform class inference that assigns a probabilistic label to every unlabeled instance. This turns out to be a challenging task due to multiple reasons: (1) unlike the discrete values produced by labeling functions in Snorkel, our affinity scores are real values, which introduce modelling difficulties; and (2) a higher affinity score between an instance and a prototype merely suggests that they are likely to belong to the same class, but does not actually indicate which class they actually belong to.

  To address these challenges, GOGGLES proposes a generative model that models affinity scores generated by different prototypes conditioned on different classes, and formulate the class inference problem as a maximum likelihood estimation problem. GOGGLES then uses expectation-maximization to iteratively refine the class assignments until convergence. Figure 1 shows that the two distributions for the affinity scores of two classes are clearly separated, which allows us to achieve high labeling accuracy.

On the Caltech-UCSD Birds-200-2011 dataset and Animals with Attributes 2 dataset, labels generated by GOGGLES show an average accuracy of 98.09% and 92.66% for the binary labeling task, and 92.93% and 83.44% for the multi-class labeling task respectively. Compared to the state-of-the-art data programming system Snorkel, GOGGLES provides 14.88% average improvement in terms of the quality of labels generated for the binary labeling task.

The rest of the paper is organized as follows. We discuss the formal problem of training data generation and our proposed affinity coding paradigm in Section 2. We show how GOGGLES provides a way to generate affinity scores for image datasets in Section 3. We present an expectation-maximization based algorithm for performing class label inference using affinity scores in Section 4. We present the experimental evaluations in Section 5. We discuss the related work in Section 6. We conclude and present the future work in Section 7. The GOGGLES system is open-sourced at `https://github.com/chu-data-lab/GOGGLES/`.

2. PRELIMINARY

We formally state the problem of automatic training data generation in Section 2.1. We then introduce *affinity coding*,

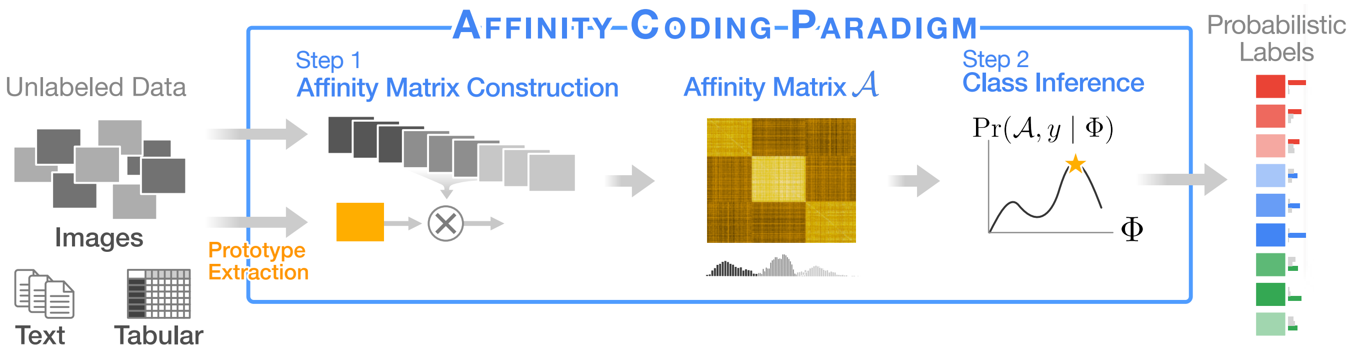

Figure 2: Affinity coding paradigm for automatic training data generation. The user defines a way to extract prototypes (rich representative features) from image, text or tabular data, and defines an affinity function between prototypes and data instances. Affinity coding leverages this information to discover similar data instances and produce probabilistic class labels that can be used for training downstream models.

a new paradigm for addressing the automatic training data generation problem in Section 2.2.

2.1 Problem Setup

In traditional supervised classification applications, the goal is to learn a classifier h_θ based on a labeled training set (x_i, y_i), where $x_i \in \mathcal{X}_{train}$ and $y_i \in \mathcal{Y}_{train}$. The classifier is then used to make predictions on a test set.

In our setting, we do not assume access to any labeled training data, namely, we only have $\mathcal{X}_{train}$ and no $\mathcal{Y}_{train}$. Let N denote the total number of unlabeled data points, and let y_i^* denote the unknown true label for x_i. Our goal is to assign a *probabilistic label* $\tilde{y}_i^k$ for every $x_i \in \mathcal{X}_{train}$, where $\tilde{y}_i^k = \Pr\big([y_i^* = k]\big) \in [0, 1]$, where $k \in \{1, 2, \ldots, K\}$ with K being the number of classes in the labeling task, and $\sum_k \tilde{y}_i^k = 1$.

These probabilistic labels can then be used to train any downstream ML models, such as a convolutional neural network (CNN) for image classification, or a long-short term memory network for text classification tasks. These probabilistic labels can be leveraged in several ways. For example, we can generate a discrete label according to the highest $\tilde{y}_i^q$ for every instance x_i. Another more principled approach is to use the probabilistic labels directly in the loss function $l(h_\theta(x_i), y)$, i.e., the expected loss with respect to $\tilde{y}$:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} E_{y \sim \tilde{y}_i}[l(h_\theta(x_i), y)] \tag{1}$$

It has been shown that as the amount of unlabeled data increases, the generalization error of the model trained with probabilistic labels will decrease at the same asymptotic rate as traditional supervised learning models do with additional hand-labeled data [Ratner et al., 2016]. In this work, however, we only focus on producing probabilistic labels.

2.2 The Affinity Coding Paradigm

We propose *affinity coding*, a completely new paradigm for obtaining high quality sources of weak supervision. The core premise of affinity coding is that *instances belonging to the same class share certain similarities, which we call affinity, that instances belonging to different classes do not have.* Figure 2 shows the affinity coding framework for generating probabilistic training data, having two main components: affinity matrix construction and class inference.

Step 1: Affinity Matrix Construction. As the raw features of an instance x_i may not be informative for computing affinity scores, the first step for affinity matrix construction involves extracting *prototypes* from input instances. Each unlabeled data point x_i in the unlabeled data can yield one or more prototypes, each of which is used as a weak supervision source ρ_j. Given N unlabeled instances and M prototypes, we compute an *affinity score* $s_i^j \in [0, 1], \forall i \in [1, N], j \in [1, M]$ between the unlabeled instance x_i and the prototype ρ_j. The output is thus represented as the affinity matrix $\mathcal{A}$ storing all the affinity scores.

Step 2: Class Inference. Each column of the affinity matrix $\mathcal{A}$ provides a weak supervision source suggesting the class membership of unlabeled points. Therefore, we have M different weak supervision sources, each potentially providing conflicting information about the class membership of each of the N data points. Thus, the job of the class inference component is to de-noise $\mathcal{A}$ and produce a probabilistic label $\tilde{y}_i$ for each data point x_i. This is done by maximizing $\Pr\big(\mathcal{A}, \tilde{y}|\Phi\big)$, where Φ denotes the parameters of a probabilistic model that is used to model $\mathcal{A}$.

The affinity coding paradigm offers a general framework for training data generation. Users only need to develop a technique for constructing the affinity matrix, and they can be reused later for similar training data generation tasks. In Section 3 and Section 4, we present a specific instantiation of the paradigm, which we call GOGGLES, that labels image datasets. We show in the experiments that the same approach is useful for labeling two different image datasets with completely different class semantics.

3. CONSTRUCTING AFFINITY MATRIX FOR IMAGES

In this section, we describe the affinity coding step of GOGGLES, a system that automatically generates labels for image datasets. As discussed before, our affinity coding paradigm is based on the proposition that examples belonging to the same class should have certain similarities. For image datasets, we hypothesize that *images from the same class would contain certain visually grounded features which are richly discriminative when compared with images of another class.*

However, it is nontrivial to design affinity scores based on visually grounded features for images due to multiple reasons: (1) raw pixel values are excessively noisy and have high variance even for images of the same class, (2) arbitrary signals that encode the image *as a whole* may not be class-discriminative since the underlying higher-order features may be localized only in specific regions of the image; and most importantly, (3) discriminative features may not be known *a priori*, in the absence of any class labels.

To alleviate these complications, GOGGLES leverages convolutional neural networks (CNNs) to transplant the data representation from the raw pixel space to a semantic space, which makes it more tractable to identify higher-order discriminative features. It has been shown that the intermediate layers of a trained CNN are able to encode human-interpretable concepts, such as edges and corners in initial layers; and textures, objects and complex patterns in the final layers [Zeiler and Fergus, 2014].

In Section 3.1, we show how we take advantage of this inherent property of trained CNNs in order to automatically identify spatially localized class-definite "concept prototypes" from the given set of images. In Section 3.2, we show how we compute the affinity scores between every image x_i and every concept prototype ρ_j. These affinity scores act as a proxy for class-discriminative signals without requiring any class labels.

3.1 Extracting Prototypes

To extract signals for labeling the images in a dataset as different classes, we need to look at different regions of each image instead of encoding the image as a whole. This is based on the intuition that the higher-order concept that we are trying to identify may be spatially localized in the image. For example, if we are given an image of a tiger, it may contain a tiger's head in one corner and rest of the image contains the background of a forest. If we are trying to extract signals by looking at the complete image, the background may add noise to the signal. Therefore we propose a technique to automatically discover *patches* in each image that may contain class-descriptive signals. These image patches in the pixel space are represented by "concept prototypes" in a semantic subspace. For each image, we try to extract K such prototypes that correspond to K different patches in the pixel space.

Extracting all prototypes. Let us now formally define our approach. To begin, we pass an image x_i through a series of convolutional, activation and pooling layers of a CNN to obtain $F_i = f(x_i)$, as illustrated in Figure 3. F_i is also called a "filter map", and has dimensions $C \times H \times W$, where C, H and W are the number of channels, height and width of the filter map respectively. Let us also denote indexes over the height and width dimensions of F_i with h and w respectively. Hence, each vector $v_i^{(h,w)} \in \mathbb{R}^C$ (spanning the channel axis) in the output volume F_i can be backtracked to a rectangular patch at the corresponding location in the input image x_i. The location of the patch is determined by computing the gradients of $v_i^{(h,w)}$ with respect to the input pixels. All pixels which are found to have non-zero gradients are a part of this patch. This region in the image is formally known as the *receptive field* of $v_i^{(h,w)}$, which is typically a continuous region. Since any change in this patch will induce a change in the vector $v_i^{(h,w)}$, we say that $v_i^{(h,w)}$ encodes the semantic concept present in the patch.

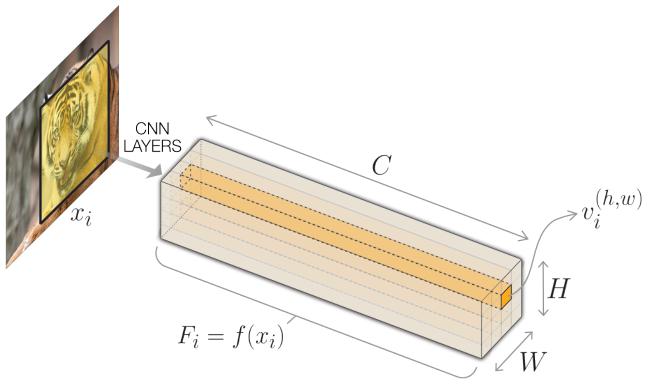

Figure 3: Representation of an image patch in semantic space. Here, we show the output volume view of a filter map. An image x_i is passed through a CNN to obtain the filter map F_i that has dimensions $C \times H \times W$. Since the yellow rectangular patch highlighted in the image induces the activation values of the vector $v_i^{(h,w)}$, which is shown in orange, we say that the vector $v_i^{(h,w)}$ is the semantic representation of the abstract visual concept contained in the patch. Hence, concepts contained in different regions of the image in the pixel space are represented in the semantic space by the corresponding vectors spanning the C dimension of the filter map. For example, in this figure, the highlighted $v_i^{(h,w)}$ represents a "tiger's head" concept.

Example 1. *Suppose we obtain a filter map from the CNN model having dimensions* $C(=512) \times H(=7) \times W(=7)$. *There are a total of* $H \times W = 49$ *possible concept prototypes of dimension* 512×1.

Selecting top K best prototypes. In an image x_i, obviously not every patch and the corresponding encoded semantic concept $v_i^{(h,w)}$ is a good signal. In fact, many patches in an image correspond to background noise that are uninformative for labeling an image. Therefore, we need a way to intelligently select the top K most informative semantic prototypes from all the $H \times W$ possible ones.

In this regard, we perform a 2D Global Max Pooling operation (GMP) across each channel of F_i to determine the top K channels containing the highest magnitude of activation. That is, we reduce F_i to $C \times 1 \times 1$ by doing a max operation over the $H \times W$ dimensions for each channel. We use this $C \times 1 \times 1$ vector containing the highest activation value of each channel to get the top K activated channels c_k, where $k \in \{1, \ldots, K\}$. Since these K channels are most activated, we hypothesize that they capture class-specific abstract concepts present in the image. Consequently, for each maximally activated channel c_k, we identify the concept prototype $v_{ik} = v_i^{(h,w)}$ if it contains the highest activation value for the channel c_k. More concretely, we define the k^{th} concept prototype for the i^{th} image as,

$$v_{ik} = v_i^{(h,w)} \ni h, w = \underset{h,w}{\operatorname{argmax}} F_i[c_k; h; w]. \qquad (2)$$

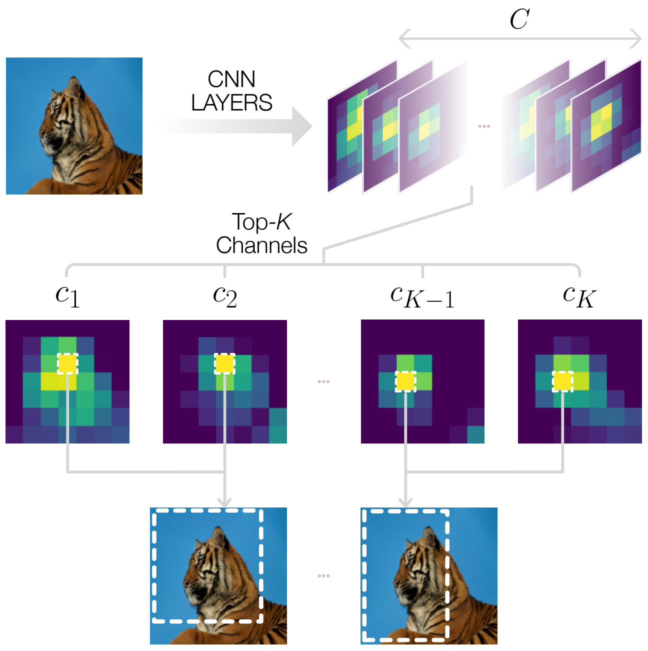

Figure 4: Extracting prototypes from the semantic space. We can represent a filter map as a stack of C channels, as shown in the top row. We select the top-K channels having the highest activation values. Next, we pick the highest value in each of these top-K channels, as indicated by the white dotted squares in the middle row. Each concept prototype corresponds to the vector of dimensions $C \times 1$ that spans across the channels and contains the highest channel activation value for each of these channels. This can then be backtracked to a patch in the input image, as shown in the bottom row. Notice here that the location of the maximum activation value for c_1 and c_2 (or c_{K-1} and c_K) are the same. Hence, they correspond to the the same concept prototype vector, and consequently, the same image patch.

Note that the pair (h, w) may not be unique across the channels, yielding the same concept prototypes. Hence, we drop the duplicate $v_i^{(h,w)}$'s and only keep the unique prototypes. This approach is illustrated in Figure 4.

Doing this for each image in the dataset, we can build a collection $\boldsymbol{\rho}$ of prototypes ρ^j, where,

$$\boldsymbol{\rho} = \{\rho^j : \rho^j = v_{ik} \ \forall i, k\}. \tag{3}$$

Here, each prototype is represented as a vector ρ^j in the semantic space, but also corresponds to a patch in the pixel space, as shown in Figure 3. We can see that by building semantic prototypes this way, we can immediately obtain many diverse signals (as many as KN) that are useful for determining image labels.

3.2 Computing Affinity

Having extracted concept prototypes for each image, next we aim to identify other images containing similar concepts. To this end, we construct an affinity matrix $\mathcal{A} \in \mathbb{R}^{N \times M}$, where N is the number of images in the dataset and M is the total number of concept prototypes extracted from all the images. Since we drop duplicate prototypes, we have $M \leq KN$. Each entry s_i^j in the matrix $\mathcal{A}$ is the affinity score between the i^{th} image x_i and the j^{th} prototype ρ^j. To compute the affinity between an image and a concept prototype, we again rely on the semantic space to capture this relationship. We calculate the similarity between the prototype ρ^j and the vector $v_i^{(h,w)} \forall (h, w) \in \{(1, 1), \ldots, (H, W)\}$ contained in $F_i = f(x_i)$ using a similarity function $s(\cdot)$, and pick the highest value as the affinity score s_i^j. That is,

$$s_i^j = \max_{h,w} s(\rho^j, v_i^{(h,w)}). \tag{4}$$

In other words, this approach tries to find the "most similar patch" in each image x_i with respect to the patch corresponding to the prototype ρ^j. For example, let's say a prototype ρ_j corresponds to a patch containing a tiger's head, then the j^{th} column in the affinity matrix $\mathcal{A}$ should intuitively have a high value in the i^{th} row if the image x_i also contains *some* patch with a tiger's head; and correspondingly, a low value in the i^{th} row if the image x_i does not contain that concept. This is illustrated in Figure 5, where there is a clear separation between between the class images based on affinity scores assigned by the prototypes. Hence, this technique would cluster images having similar visually grounded semantic features by assigning high affinity scores to prototype-image pairs of the same class, and low affinity scores to prototype-image pairs of different classes. Since we can also backtrack from ρ^j as well as $v_i^{(h,w)}$ to patches in the image, this approach has the added advantage of being interpretable. A human can visually inspect the concept prototypes in the pixel space as well as the image patches that induce high affinity scores in order to potentially refine the results.

For computing the affinity, we use the cosine similarity metric as the similarity function $s(\cdot)$:

$$s(a, b) = \frac{a^T b}{\|a\|_2 \|b\|_2}. \tag{5}$$

If the CNN model uses ReLU activation, all values of F_i are non-negative. Hence, using the cosine similarity metric, we get affinity scores s_i^j in the range $[0, 1]$.

Discussion. In our particular instantiation of the GOGGLES pipline, we use a pre-trained VGG-16 model [Simonyan and Zisserman, 2014], and take the output from the last 2D max-pooling layer to parameterize $f(\cdot)$, with $K = 5$. This choice is based on the intuition that VGG-16, which has been trained for the classification task on the ImageNet dataset, can encode higher-order semantic abstractions in its last layer even if we show it images from a domain it was *not* trained on. However, the underlying paradigm of affinity coding which we introduce in this work is not tied to any particular CNN architecture. This choice can in-fact be replaced by unsupervised CNN models such as autoencoders that can be trained to reproduce the images we want to label, which we leave for future work.

In summary, our approach automatically identifies semantically meaningful prototypes from the dataset, and leverages these prototypes for providing proxy discriminative signals in the form of affinity scores. However, these signals may be noisy and conflicting. In the next section, we propose a probabilistic approach to denoise these signals and learn probabilistic labels for the dataset.

4. CLASS INFERENCE

In this section, we describe GOGGLES' class inference module. For the most part of our discussion, we focus on the binary labeling task for clarity. We also show how we extend our method for the multi-class labeling task at the end of this section.

Given the affinity matrix $\mathcal{A}$, where each row i represents an unlabelled data point and each column j represents a prototype extracted from these data points, GOGGLES proceeds to assign a probabilistic label $\tilde{y}_i$ for every example x_i. On the surface, the affinity matrix $\mathcal{A}$ produced by GOGGLES is similar to the labeling matrix Λ produced by Snorkel [Ratner et al., 2017], in the sense that each column of $\mathcal{A}$ can be seen as a weak supervision signal and the goal is to de-noise these signals for label inference. However, doing class inference on $\mathcal{A}$ is more challenging than de-noising Λ for the following reasons:

- The entries in Λ are discrete values from $\{-1, 1, 0\}$, while the affinity scores produced by the GOGGLES framework are continuous values in the range $[0, 1]$. This means that any probabilistic models used for class inference need to handle learning and inference efficiently on continuous variables. The factor graph model employed by Snorkel assumes discrete variables and cannot be directly adapted to handle continuous variables. At the same time, our continuous affinity scores are actually more expressive in modelling weak signals than discrete scores, and if de-noised properly, should produce higher quality labels, as we show in Section 5.

- The value in Λ provides a direct signal to the class assignment. For example, $\Lambda[i, j] = 1$ is a direct signal from weak supervision source j suggesting that $y_i = 1$. On the other hand, a high value of s_i^j (e.g., 0.99) in $\mathcal{A}$ does not give direct indication to the label assignment of y_i — it only suggests that x_i is likely to belong to the same class of the example, from which the prototype ρ_j was derived. This additional uncertainty is a complication due to the affinity matrix design that Snorkel does not need to handle.

These difficulties may lead up to the notion of turning to unsupervised clustering methods such as k-means clustering, but we follow a more principled approach [Raykar et al., 2009] to resolving the uncertainty over the supervision sources. We recognize that the class membership of each prototype is implicitly tied to the class label of the data point that the prototype is derived from. We denote this class membership of the j^{th} column with z_j, and propose a probabilistic framework for inferring the class label y_i for each data point x_i. We express our proposed formulation for binary classification first, and then extended it to the multi-class case as well.

In the following, we first introduce a generative model in Section 4.1 that models every prototype ρ_j in order to handle the above difficulties. Based on the generative model, we formulate the label inference problem as a maximum likelihood estimation problem in Section 4.2. We propose an expectation-maximization approach for solving the problem in Section 4.3.

4.1 A Generative Model for Prototypes

For every prototype ρ_j, we introduce a generative model that generates the affinity scores s_i^j for all images x_i. Let $s^j \in [0, 1]$ be a variable denoting the affinity score between an instance x and the prototype ρ_j. Let $y \in \{0, 1\}$ be the actual unobserved class label for x. Each prototype provides some information about the hidden true class label y. If the true label is one, namely, $y = 1$, then s^j follows a Gaussian distribution parameterized by $\mathcal{N}(\mu_{j1}, \sigma_{j1}^2)$. If the true label is zero, namely, $y = 0$, then s^j follows another Gaussian distribution parameterized by $\mathcal{N}(\mu_{j0}, \sigma_{j0}^2)$. Putting them in together, we formulate the generative model for s^j using two Gaussians as follows:

$$s^j \sim (1 - y)\mathcal{N}(\mu_{j0}, \sigma_{j0}^2) + y\mathcal{N}(\mu_{j1}, \sigma_{j1}^2). \tag{6}$$

As discussed earlier, a complication we have is that we do not know the class membership of each prototype ρ_j, which we denote as z_j. If $z_j = 1$, then we expect $\mathcal{N}(\mu_{j1}, \sigma_{j1}^2)$ to have a higher mean and smaller variance than $\mathcal{N}(\mu_{j0}, \sigma_{j0}^2)$; otherwise, we expect the opposite. Luckily, this complication can be circumvented by using the EM formulation (c.f. Section 4.3). This is because EM is an iterative algorithm, and at every iteration, we have an assignment of labels y_i for all instances $x_i, \forall i \in [1, N]$. As all the prototypes are derived from the instances, we also have an assignment for $z_j, \forall j \in [1, M]$. Therefore, the z_j's are implicitly tied to the y_i's, and we do not need to model uncertainty over z_j separately.

Given the generative model s^j for prototype ρ_j, a current assignment of class labels y_i (and hence z_j), the question is what is the probability of observing a particular value s_i^j, denoted as $\Pr(s_i^j | y_i)$. Depending on the assignment of y_i's, we give the calculation equations as follows:

$$\begin{aligned} \alpha_i^j &= \Pr(s_i^j | y_i = 1) = \Pr(s_i^j | y_i = 1, z_j) \\ &= \left[CDF_{j1}(s_i^j)\right]^{z_j} \left[1 - CDF_{j1}(s_i^j)\right]^{(1 - z_j)}, \end{aligned} \tag{7}$$

$$\begin{aligned} \beta_i^j &= \Pr(s_i^j | y_i = 0) = \Pr(s_i^j | y_i = 0, z_j) \\ &= \left[CDF_{j0}(s_i^j)\right]^{(1 - z_j)} \left[1 - CDF_{j0}(s_i^j)\right]^{z_j}, \end{aligned} \tag{8}$$

where, CDF_{j1} is the cumulative distribution function (CDF) of $\mathcal{N}(\mu_{j1}, \sigma_{j1}^2)$ and CDF_{j0} is the CDF of $\mathcal{N}(\mu_{j0}, \sigma_{j0}^2)$.

Example 2. *Figure 5 shows the Gaussian distributions learned by our approach on the given affinity matrix. Let us consider the case when $z_j = 0$. This corresponds to the affinity scores marked as Class 0 Prototypes in the figure, and correspondingly, the histogram shown on the left. Following (7) and (8), α_i^j corresponds to the the area to the right of the observed score (1 - CDF_{j1}) for the Gaussian distribution shown in orange, and β_i^j corresponds to the the area to the left of the observed score (CDF_{j0}) for the Gaussian distribution shown in blue. Hence, α_i^j will give a high probability to low scores, denoting $y_i \neq z_j = 0$, and β_i^j will give a high probability to high scores, denoting $y_i = z_j = 0$.*

4.2 Maximum Likelihood Estimation

Given the observed affinity matrix $\mathcal{A} \in \mathbb{R}^{N \times M}$, we want to estimate the parameters $\Phi = \{\boldsymbol{\mu_0}, \boldsymbol{\sigma_0^2}, \boldsymbol{\mu_1}, \boldsymbol{\sigma_1^2}\}$ that maximizes the data likelihood, where $\boldsymbol{\mu_0}$ (and the rest three bold parameters) denotes a vector of parameters including

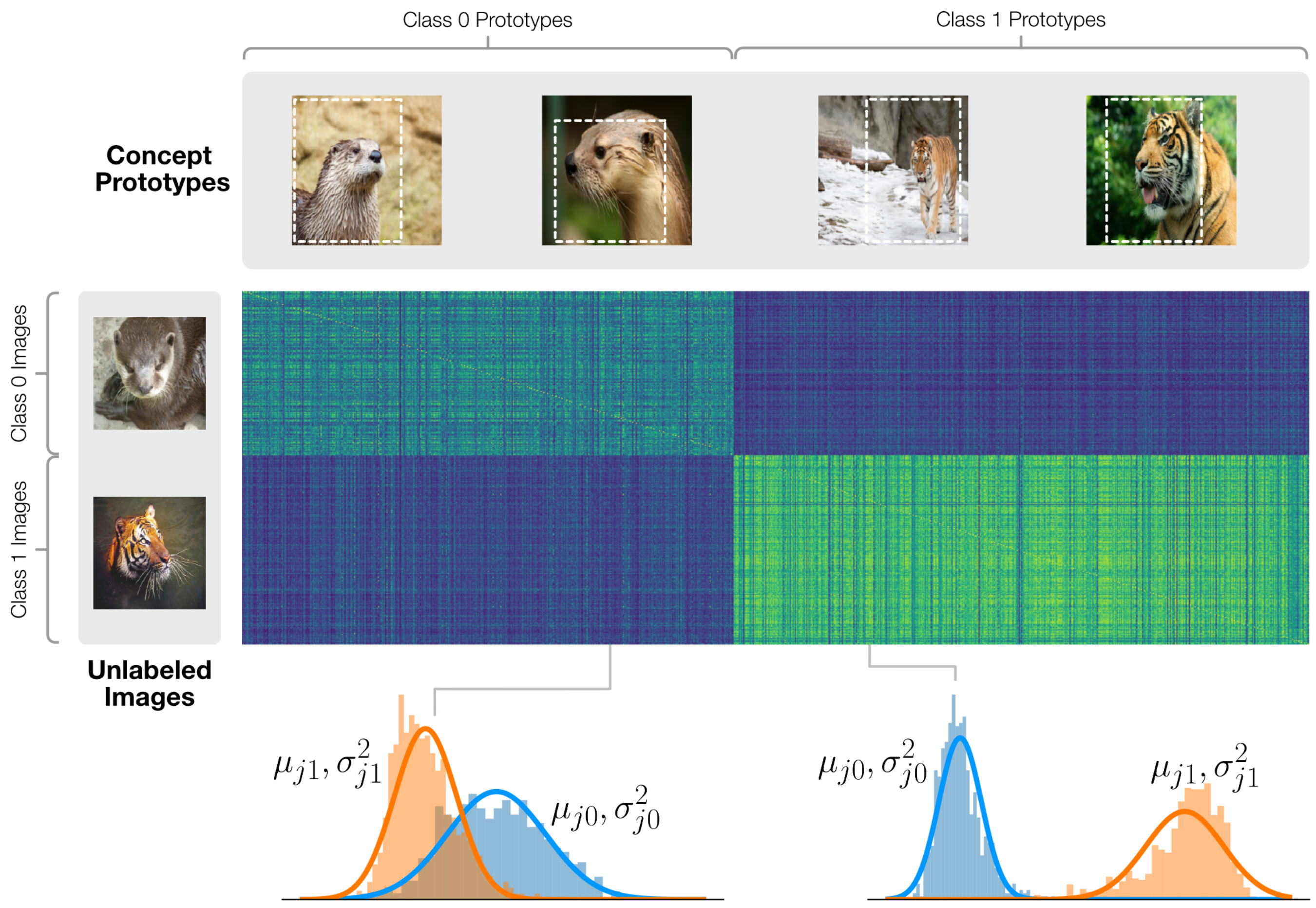

Figure 5: Affinity Matrix. This figure illustrates an affinity matrix $\mathcal{A}$ of dimensions 1635×4938 computed for classes 13 (otter) and 36 (tiger) from the AwA2 dataset. Here, higher affinity scores are shown in green and lower affinity scores are shown in blue. Each row in the affinity matrix represents an unlabeled data point x_i, and each column represents a concept prototype ρ_j derived from the data points. The rows and columns are sorted by class only for visual intuition. This figure shows that images belonging to a particular class have relatively high affinity scores with prototypes of the same class, and correspondingly low scores with prototypes of another class, i.e., otter prototypes assign high scores to otter images, and low scores to tiger images. The histograms shown in the bottom row represent the distribution of the scores conditioned on the class labels. The parameters for these distributions are learned by our probabilistic approach as defined in Section 4. Notice here that some prototypes (in the column space) are more noisy, and hence have highly overlapping distributions. From our formulation, the effect of these noisy prototypes are negated in our class inference phase and the probabilistic labels are more influenced by the well separated distributions.

$\mu_{j0}, \forall j \in [1, M]$. Given that all instances are independent, the likelihood function can be written as follows:

$$\Pr(\mathcal{A}|\Phi) = \prod_{i=1}^{N} \Pr(s_i^1, \ldots, s_i^M|\Phi). \tag{9}$$

Conditioning on the true label, we get,

$$\Pr(\mathcal{A}|\Phi) = \prod_{i=1}^{N} \Big\{ \Pr(s_i^1, \ldots, s_i^M|y_i = 1, \boldsymbol{\mu_1}, \boldsymbol{\sigma_1^2}) \Pr(y_i = 1) + \Pr(s_i^1, \ldots, s_i^M|y_i = 0, \boldsymbol{\mu_0}, \boldsymbol{\sigma_0^2}) \Pr(y_i = 0) \Big\}. \tag{10}$$

We use the loose assumption that the prototypes are independent given a label. This assumption is mostly true as most prototypes are derived from different instances. Therefore, we define,

$$a_i = \Pr(s_i^1, \ldots, s_i^M|y_i = 1, \boldsymbol{\mu_1}, \boldsymbol{\sigma_1^2}) = \prod_{j=1}^{M} \Pr(s_i^j|y_i = 1, \mu_{j1}, \sigma_{j1}^2) = \prod_{j=1}^{M} \alpha_i^j. \tag{11}$$

Similarly,

$$b_i = \Pr(s_i^1, \ldots, s_i^M|y_i = 0, \boldsymbol{\mu_0}, \boldsymbol{\sigma_0^2}) = \prod_{j=1}^{M} \Pr(s_i^j|y_i = 0, \mu_{j0}, \sigma_{j0}^2) = \prod_{j=1}^{M} \beta_i^j. \tag{12}$$

Hence,

$$\Pr(\mathcal{A}|\Phi) = \prod_{i=1}^{N} \{a_i p + b_i(1-p)\}, \tag{13}$$

where we define $p = \Pr(y = 1)$, which captures the percentage of examples having class label 1.

We find the maximum likelihood estimator by maximizing the log-likelihood:

$$\hat{\Phi}_{\text{MLE}} = \{\hat{\boldsymbol{\mu_0}}, \hat{\boldsymbol{\sigma_0^2}}, \hat{\boldsymbol{\mu_1}}, \hat{\boldsymbol{\sigma_1^2}}\} = \operatorname*{argmax}_{\Phi} \ln \{\Pr(\mathcal{A}|\Phi)\} \tag{14}$$

Given $\hat{\Phi}_{\text{MLE}}$, the probabilistic label for x_i is thus $\tilde{y}_i = \Pr(y_i = 1|s_i^1, ..., s_i^M, \hat{\Phi}_{\text{MLE}})$

4.3 Expectation Maximization

We can compute the maximum likelihood estimate $\hat{\Phi}_{\text{MLE}}$ using the Expectation Maximization (EM) technique [Dempster et al., 1977], by treating the class labels $\boldsymbol{y} = \{y_1, \ldots, y_n\}$ as the latent variables. The complete data log-likelihood can be written as:

$$\ln \Pr(\mathcal{A}, \boldsymbol{y}|\Phi) = \sum_{i=1}^{N} y_i \ln p_i a_i + (1 - y_i) \ln (1 - p_i) b_i \tag{15}$$

Each iteration of the expectation maximization algorithm comprises of an (E)xpectation step and a (M)aximization step. In the E-step, we maximize the complete data log-likelihood given the current model parameters Φ to find an estimate of the true label for each image. In the M step, given the label estimates we recompute the model parameters. These two steps are performed iteratively until the estimates converge, i.e., the estimated labels do no change any more.

1. **E Step.** Given the current estimate of model parameters Φ and the affinity matrix $\mathcal{A}$, the expectation of the log-likelihood is computed as

   $$\mathbb{E}\{\ln \Pr(\mathcal{A}, \boldsymbol{y}|\Phi)\} = \sum_{i=1}^{N} \gamma_i \ln p_i a_i + (1 - \gamma_i) \ln (1 - p_i) b_i \tag{16}$$

   ,where $\gamma_i = \Pr(y_i = 1|s_i^1, ..., s_i^M, \Phi)$.

   Using Bayes theorem we can say that:

   $$\begin{aligned} \gamma_i &\propto \Pr(s_i^1, ..., s_i^M | y_i = 1, \Phi) \Pr(y_i = 1) \\ &= \frac{a_i p}{a_i p + b_i(1-p)} \end{aligned} \tag{17}$$

2. **M Step.** Given the probabilistic labels $\boldsymbol{\gamma}$ and the affinity matrix $\mathcal{A}$, the model parameters Φ are then re-computed by maximizing (16).

   Finding an analytic solution of Φ that maximizes (16) is challenging. Hence, we perform random sampling to get discrete labels for every image, where each image x_i has a probability of γ_i of being assigned label 1. Given the discrete assignments of labels, maximizing (16) becomes equivalent to maximizing (14), with p being the percentage of images that have label 1 in the assignments. We then fit the two normal distributions according to the two sets of instances, depending on the discrete labels, to update our parameters Φ.

Initialization of the EM process. The EM process needs an initialization, i.e., an initial guess of y_i. We perform a k-means clustering (2-means clustering for binary class inference) on the examples x_i, using the i^{th} row in the $\mathcal{A}$ matrix as the features for x_i. The assumption is that, for two images x_{i_1} and x_{i_2}, if they belong to the same class, then they are more likely to share similar values in $s_{i_1}^j$ and $s_{i_2}^j$, for every prototype ρ_j, regardless of which class ρ_j was derived from.

4.4 Multi-class Labeling

We describe how we extend the aforementioned approach for assigning class memberships for multi-class labeling tasks. Let K denote the number of classes (labels) in our labeling task. We still need to have a generative model for s^j, except now we need to have K different Gaussian distributions depending on the hidden true label of an instance, that is,

$$s^j \sim \mathcal{N}(\mu_{jk}, \sigma_{jk}^2) \quad \text{if} \quad y = k. \tag{18}$$

The probability of observing a particular value s_i^j, given the current assignment of y_i (and z_j), is thus calculated as follows:

$$\begin{aligned} \alpha_i^{jk} &= \Pr(s_i^j | y_i = k) = \Pr(s_i^j | y_i = k, z_j) \\ &= [CDF_{jk}(s_i^j)]^{\mathbb{1}(z_j = k)} [1 - CDF_{jk}(s_i^j)]^{\mathbb{1}(z_j \neq k)} \end{aligned} \tag{19}$$

where CDF_{jk} is the cumulative distribution function (CDF) of $\mathcal{N}(\mu_{jk}, \sigma_{jk}^2)$ and $\mathbb{1}$ is the indicator function.

Given the new generative model, we can thus define a_i^k as follows:

$$\begin{aligned} a_i^k &= \Pr(s_i^1, \ldots, s_i^M | y_i = k, \boldsymbol{\mu_k}, \boldsymbol{\sigma_k^2}) \\ &= \prod_{j=1}^{M} \Pr(s_i^j | y_i = k, \mu_{jk}, \sigma_{jk}^2) = \prod_{j=1}^{M} \alpha_i^{jk}. \end{aligned} \tag{20}$$

Hence, the new likelihood function is:

$$\Pr(\mathcal{A}|\Phi) = \prod_{i=1}^{N} \{ \sum_{k=1}^{K} a_i^k p^k \}, \tag{21}$$

where p^k is the percentage of instances having label k and $\sum_{k=1}^{K} p^k = 1$.

The EM algorithm for maximizing the new likelihood function mostly stays the same. For the initial class assignments, we run the k-means clustering algorithm with the number of clusters equal to K.

Discussion. The class inference module technically produces a clustering of the unlabeled instances, where each cluster corresponds to a class in the multi-class labeling task. However, we do not automatically know which class label a particular cluster corresponds to. We assume that a domain expert will look at each cluster and decide the cluster-to-class mapping, which should be a very simple task.

5. EXPERIMENTS

We compare GOGGLES, our system for implementing the affinity coding paradigm, to the Snorkel system that implements the data programming paradigm. The main points we seek to validate in this section include: (1) the end-to-end labeling accuracy of GOGGLES for the binary and multi-class labeling tasks; (2) comparing the weak supervision sources

Dataset	Class #	Dataset Size	VGG-16	LF+PGM (Snorkel)	LF+EM	AF+PGM	AF+KM	AF+EM (GOGGLES with random initialization)	AF+EM (GOGGLES)
	5,86	104(44,60)	-	0.8077	0.9231	**0.9904**	0.9808	0.9663	0.9808
	75,105	106(57,49)	-	0.8396	0.9717	0.9906	**1.0000**	0.9764	**1.0000**
	8,92	108(48,60)	-	0.7778	0.7037	0.5370	**0.9167**	0.8796	**0.9167**
	76,101	110(60,50)	0.9545	0.9818	0.9818	0.7727	**0.9909**	0.9682	**0.9909**
CUB	7,181	112(53,59)	-	0.9821	0.5268	0.7634	**1.0000**	**1.0000**	**1.0000**
	71,119	119(60,59)	-	0.8739	0.8824	0.5126	**0.9832**	0.8697	**0.9832**
	21,132	120(60,60)	-	0.9583	0.5000	0.7458	**0.9917**	**0.9917**	**0.9917**
	81,114	120(60,60)	-	0.9083	0.8917	0.5083	0.9750	0.9583	**0.9792**
	102,136	120(60,60)	-	0.9083	0.8667	0.7500	0.9667	0.9625	**0.9750**
	127,143	120(60,60)	-	0.5000	0.5083	0.5083	**0.9917**	0.9750	**0.9917**
		Average	0.9545	0.8538	0.7756	0.7079	0.9796	0.9547	**0.9809**
	30,35	655(383,272)	0.1496	-	-	**0.9817**	0.6718	0.8977	0.6802
	10,35	772(500,272)	0.5699	-	-	**0.9650**	0.8847	0.8925	0.8912
	2,12	952(852,100)	0.7826	-	-	**0.9296**	0.8414	0.6381	0.8256
	35,37	1167(272,895)	0.0865	-	-	×	**0.9794**	0.9734	0.9786
AwA2	32,41	1219(589,630)	0.7719	-	-	0.5119	**0.9352**	0.9282	0.9286
	20,42	1585(872,713)	0.8479	-	-	×	0.9842	0.9861	**0.9880**
	13,36	1635(877,758)	0.9168	-	-	×	0.9951	**0.9976**	0.9963
	38,42	1883(1170,713)	0.8959	-	-	×	0.9995	**1.0000**	**1.0000**
	21,27	1928(728,1200)	0.7433	-	-	×	0.9860	0.9844	**0.9870**
	8,23	2453(1033,1420)	-	-	-	×	0.9894	0.9898	**0.9902**
		Average	0.5764	-	-	0.8470	0.9267	**0.9288**	0.9266

Table 1: Results for the binary labeling task. The $\times$ symbol indicates timeout after 24 hours. The - symbol indicates setting that are not applicable.

used for labeling, i.e., labeling functions in Snorkel v.s. affinity matrix in GOGGLES; and (3) comparing the class inference component for de-noising, i.e., the graphical model based approach in Snorkel v.s. the EM-based approach in GOGGLES.

5.1 Experiment Setup

Datasets. We use the following two datasets for our experimental evaluation.

- Caltech-UCSD Birds-200-2011 (CUB) dataset [Wah et al., 2011]: The CUB dataset comprises of 11,788 images of 200 bird species. Each image is also provided with image level attribute information like white head, grey wing etc. The attribute annotations help explain the visual characteristics of each image. We treat these human annotations as a proxy for labeling functions for evaluating Snorkel.
- Animals with Attributes 2 (AwA2) dataset [Xian et al., 2018]: The AwA2 dataset comprises of 37322 images of 50 animal species with 85 class level attribute annotations.

For binary labeling tasks, we randomly pick 10 pairs of classes from both the datasets. For multi-class labeling tasks, we randomly pick 3 sets of 5 species for both the datasets and show the results for the first 3, 4, and all classes in the set.

Competing Methods for binary labeling tasks. Both Snorkel and GOGGLES have two components: (1) constructing a matrix that includes all the weak supervision sources. Snorkel uses the labeling function based approach (LF), and GOGGLES uses the affinity matrix (AF) based approach. (2) performing class inference using the constructed matrix. Snorkel uses the probabilistic graphical model based approach (PGM), and GOGGLES uses the expectation-maximization based approach (EM). To compare the effects of these components on labeling performance, we construct and compare the following methods for training data generation.

- VGG-16 as a labeler: Since we use VGG as our CNN model to extract the concept prototypes, we compare the labeling accuracy of VGG-16 with our algorithm. For both binary and multi-class labeling tasks, if all the classes in the set are present in the ImageNet dataset which the model was trained on, we report the results of VGG-16.
- LF + PGM (Snorkel). This is Snorkel's approach. However, Snorkel needs humans to write the LFs. Luckily, CUB dataset contains human annotations with respect to multiple class properties (e.g., whether an image has blue tail or not), and it also contains which class a particular property indicates. Hence, we use the human annotations in the CUB dataset to simulate the labeling functions needed by the Snorkel denoising technique. Note that this method is not applicable to the AwA2 dataset as it does not have any human annotations.
- LF + EM: This method swaps the PGM based approach for de-noising with our EM-based approach. The purpose is to compare the de-noising performance of PGM v.s. EM for the matrix that contains weak supervision signals provided by humans.
- AF + PGM: This method tests the capabilities of using PGM for denoising the signal matrix provided by affinity coding. The PGM expects the values in the matrix to be in $\{-1, 0, 1\}$, while our affinity matrix

comprises of scores in the range $[0, 1]$. Hence, we need to discretize the scores in the affinity matrix. One hurdle here is that, we do not know the class membership of each column j *a priori*. Hence we first assign a class label to each *column j* by running the k-means algorithm over each column with the rows as features, i.e., we run the k-means algorithm on the transpose of the affinity matrix. Next, we calculate a threshold for each column by taking the mean of the minimum and maximum values in the column, and then assign the class label corresponding to the column to each row having a score greater than the column threshold. Correspondingly, we assign the label 0 (uncertain) to rows having scores less than the column threshold. We denoise the resulting matrix using the PGM provided by Snorkel and report the results.

- AF + KM: We also analyze the performance of clustering algorithm k-means (KM) as a denoising technique for the generated affinity matrix for each set of species for both the datasets. This is essentially the GOGGLES system without using the EM approach to iteratively refine the label assignments.
- AF + EM (GOGGLES): We evaluate the performance of Expectation Maximization(EM) algorithm explained in 4.3 as a denoising technique on top of the constructed affinity matrix.
- AF + EM (GOGGLES with random initialization): This is the same algorithm as GOGGLES, except that, instead of using the results of k-means as the initial class assignments of the EM algorithm, we simply do random initialization. We only include this method to verify the robustness of our EM algorithm.

Competing Methods for multi-class labeling tasks. As Snorkel does not provide implementations for multi-class labeling tasks, we only compare the AF + KM method with the AF + EM (GOGGLES) method. We also report the results for using VGG-16 as a labeler if all the classes are found in the ImageNet dataset that VGG-16 is trained on.

Evaluation Metrics. Since the final labeling results from both Snorkel and GOGGLES are probabilistic, we need to convert them into discrete labels to compare them with the ground truth labeling. For EM-based class inference methods, we simply use the discrete labels produced by the M-step in the last iteration of the EM algorithm. For PGM-based class inference methods, we follow Snorkel's recommendation: we take the average of the minimum and maximum probabilities as the threshold probability, and instances that have probabilities above the threshold are labeled 1, and otherwise 0. The final labeling accuracy is the percentage of unlabeled instances that have the same discrete labels as the ground truth.

Implementation Details. Computing the affinity matrix code is vectorized end to end using pytorch. In addition, since the class inference is probabilistic for AF + KM, AF + EM (random), AF + EM (KM) and AF + PGM we run the experiments 10 times and report the median accuracy values for these runs for each experiment. However, PGM (Snorkel) takes a lot of time to denoise the affinity matrix for AwA2 dataset and therefore, we stop the code after 24 hours and report the results for 4 class pairs for only 1 run.

5.2 Binary Labeling Tasks

Table 1 shows the labeling accuracy comparisons for all the discussed methods. There are multiple observations from the results:

- We show that GOGGLES consistently produces the best accuracy in comparison to the other algorithms. On the CUB dataset, GOGGLES produces an absolute labeling accuracy of 98.09% on an average, which is 14.88 % more than the average accuracy of Snorkel i.e. 85.38%. On the AwA2 dataset, GOGGLES produces an absolute labeling accuracy of 92.66%.
- We can see that methods that use AF are much better than methods that use LF as the weak supervision sources. This shows that our proposed affinity coding paradigm provides a much richer set of weak supervision signals as compared to data programming. This is primarily because of the affinity coding design: every unlabeled image itself can provide at least one supervision signal.
- We also compare the two class inference components: PGM v.s. EM. Comparing LF + PGM with LF + EM, we can see that there is no clear dominance in terms of de-noising human labeling functions. Sometimes PGM is better than EM, and sometimes EM is better. Comparing AF + PGM with AF + EM, we can see that EM is consistently better than PGM on the CUB dataset, while we get mixed results on the AwA2 dataset. However, note that, we get many '-' entries for AF + PGM on the AwA2 dataset, which means that PGM has exceeded our preset 24 hours running time limit.
- We have also compared AF + KM with AF + EM (GOGGLES) to verify the benefits of running EM on top of k-means. We can see that except for two cases where we see small accuracy degradation (35, 37 and 32, 41 on AwA2 dataset), EM always improves the labeling accuracy of k-means.
- We also compared AF + EM (GOGGLES) with AF + EM (GOGGLES with random initialization). We can see that they produce similar results, which suggest that our EM algorithm is able to iteratively refine class assignments and does not depend on a particular seed initialization. We choose k-means initialization as default in GOGGLES because it already provides a good starting point and hence EM can converage much faster.

5.3 Muti-class Labeling Tasks

Table 2 shows the results for multi-class labeling tasks. We observe the following:

- GOGGLES is able to achieve an average labeling accuracy of 92.93% and 83.44% on the Caltech-UCSD Birds-200-2011 dataset and Animals with Attributes 2 dataset respectively for the multiclass labeling task across randomly selected 3-class, 4-class, and 5-class labeling tasks.
- As the number of classes gets higher, the labeling accuracy usually goes down. This is expected as the labeling tasks are harder for more classes.

Dataset	Class #	Dataset Size	VGG	AF + KM	AF+EM (GOGGLES)
CUB	5,10,56	164(44,60,60)	-	0.9390	0.9451
	5,10,56,86	224(44,60,60,60)	-	0.9642	0.9687
	20,44,81	179(59,60,60)	-	0.9665	0.9665
	20,44,81,114,194	299(59,60,60,60,60)	-	0.9431	0.9565
	71,74,119	179(60,60,59)	-	0.9218	0.9218
	71,74,119,162	239(60,60,59,60)	-	0.9163	0.9205
	71,74,119,162,164	299(60,60,59,60,60)	-	0.8696	0.8261
		Average		0.9315	0.9293
AwA2	9,10,30	1576(383,174,1019)	-	0.8221	0.8391
	9,10,30,35	2076(383,174,1019,500)	-	0.7125	0.7065
	6,8,23	2493(1033,747,713)	0.3500	0.9740	0.9806
	6,8,23,42,45	4781(1033,747,713,868,1420)	0.5200	0.8611	0.8671
	22,30,32	2391(383,1344,664)	-	0.9034	0.9327
	22,30,32,35	2663(383,1344,664,272)	-	0.8149	0.7861
	22,30,32,35,40	3252(383,1344,664,272,589)	-	0.7106	0.7284
		Average		0.8284	0.8344

Table 2: Multi-class labeling results

- Comparing AF + KM with AF + EM (GOGGLES), we can see that EM provides a small amount of improvements over KM. This suggests that (1) the biggest factor for our labeling performance can be attributed to the affinity matrix and (2) EM does in fact further improve the performance of k-means.

6. RELATED WORK

In this section, we highlight how ML community tackles the challenges associated with insufficient training data. We highlight the differences between GOGGLES and data programming. We also give overviews of other related work in the data management community.

Model Training with Insufficient Data. ML community has been dealing with insufficient training data problem in a variety of ways. Active learning techniques aim at involving human labelers in a judicious way to get the maximal benefits while minimizing labeling cost [Settles, 2012]. Semi-supervised learning techniques trains models with some labeled data and a much larger set of unlabeled data [Zhu, 2005]. They usually leverage various assumptions about the data, such as smoothness and low-dimensional structure. Transfer learning uses models trained on other tasks that have many labeled data to help with training models on new tasks with less labeled data [Pan and Yang, 2010]. The end products of these approaches are the final predictive models, and hence they usually integrate the model training process and the labeling process. In contrast, we are not tied with downstream modeling, and purely aim at producing labels for the unlabeled set that can be used to train any model.

Data Programming. Data programming [Ratner et al., 2016], and the Snorkel system that implements it [Ratner et al., 2017], is a recent proposal that assists users to programmatically generate training data, and is the most relevant work to ours. In Snorkel, users need to write many labeling functions, where each labeling function provides labels for a subset of the unlabeled data. The labeling functions essentially allow users to encode any heuristic rules that may be useful for labeling data. Applying user-written labeling functions to all unlabeled data generates a labeling matrix. Given a labeling matrix, Snorkel uses the agreements and disagreements of the labeling functions on different data points to learn the accuracy and dependencies of labeling functions via factor graph models, and then produces the final probabilistic labels. Though GOGGLES and Snorkel share similar structures (GOGGLES's affinity matrix corresponds to Snorkel's labeling matrix and GOGGLES's class inference component corresponds to Snorkel's matrix de-noising component), there are multiple critical differences between them: (1) GOGGLES does not need any user input, except for the affinity functions that developers need to code. However, once coded, they can be reused for labeling future datasets; (2) Snorkel still requires a small set of labeled data (termed development set) to train the factor graph model for matrix de-noising. GOGGLES, however, requires zero labeled data, and does automatic class inference; and (3) while Snorkel uses the complicated factor graph models for matrix de-noising, GOGGLES features a much simpler EM algorithm for class inference, which surprisingly produces better results as shown in Section 5.

Related Work in Database Community. Many research problems in database community share similar technical challenges to our work. In particular,data fusion/truth discovery [Pochampally et al., 2014, Rekatsinas et al., 2017b], crowdsourcing [Das Sarma et al., 2016], and data cleaning [Rekatsinas et al., 2017a], in one form or another, all need to reconcile information from multiple sources to reach one answer. While the information sources are assumed as input in these problems, labeling training data faces the challenge of lacking enough information sources. In fact, one primary contribution of GOGGLES is the affinity coding paradigm, where each unlabeled data becomes an information source. On the other hand, Our EM-based class inference approaches are inspired by many of the reconciliations techniques proposed in database community.

Related Work in Vision Community. CNNs have been producing the state-of-the-art results for image recognition tasks, introducing further research into the discriminative power of the filter banks. Highly sparse deep layers of VGG-16 retain the most discriminative information in the representation space as opposed to AlexNet that holds background information due to larger number of max-pool layers resulting in low sparsity [Yu et al., 2016]. The DeepCluster method [Caron et al., 2018] performs discriminative learning by assigning pseudo labels using k-means clustering and then learning them in the network by optimizing weights through backpropagation with a classification loss. However, this work treats the input as a whole, whereas our work tries to identify spatially localized concepts in the input.

7. CONCLUSION AND FUTURE WORK

In this paper, we proposed affinity coding, a new paradigm for automatic generation of training data. Affinity coding is based on the proposition that instances belonging to the same class share certain similarities that instances belonging to different classes do not share. We also proposed the GOGGLES system that implements the affinity coding paradigm for image datasets. GOGGLES also includes a novel algorithm for class inference given an affinity matrix.

As far as we know, GOGGLES is the first system that performs training data generation automatically. There are many interesting followup research directions: (1) how do further increase the labeling accuracy, especially for multi-class labeling tasks; (2) can we design other (better) ways to code the affinities for images; (3) how can we apply affinity coding for generate training data for labeling tasks on other types of data, such as text and structured data.

References

M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149, 2018.

A. Das Sarma, A. Parameswaran, and J. Widom. Towards globally optimal crowdsourcing quality management: The uniform worker setting. In *Proceedings of the 2016 International Conference on Management of Data*, pages 47–62. ACM, 2016.

A. P. Davis, T. C. Wiegers, P. M. Roberts, B. L. King, J. M. Lay, K. Lennon-Hopkins, D. Sciaky, R. Johnson, H. Keating, N. Greene, et al. A ctd–pfizer collaboration: manual curation of 88 000 scientific articles text mined for drug–disease and drug–phenotype interactions. *Database*, 2013, 2013.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10): 1345–1359, 2010.

R. Pochampally, A. Das Sarma, X. L. Dong, A. Meliou, and D. Srivastava. Fusing data with correlations. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 433–444. ACM, 2014.

A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3): 269–282, 2017.

A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *Advances in neural information processing systems*, pages 3567–3575, 2016.

V. C. Raykar, S. Yu, L. H. Zhao, A. Jerebko, C. Florin, G. H. Valadez, L. Bogoni, and L. Moy. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *Proceedings of the 26th Annual international conference on machine learning*, pages 889–896. ACM, 2009.

T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment*, 10(11):1190–1201, 2017a.

T. Rekatsinas, M. Joglekar, H. Garcia-Molina, A. Parameswaran, and C. Ré. Slimfast: Guaranteed results for data fusion and source reliability. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1399–1414. ACM, 2017b.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

B. Settles. Active learning: Synthesis lectures on artificial intelligence and machine learning. *Long Island, NY: Morgan & Clay Pool*, 2012.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata. Zero-shot learning-a comprehensive evaluation of the good, the bad and the ugly. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

W. Yu, K. Yang, Y. Bai, T. Xiao, H. Yao, and Y. Rui. Visualizing and comparing alexnet and vgg using deconvolutional layers. In *Proceedings of the 33 rd International Conference on Machine Learning*, 2016.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

X. J. Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2005.