Active Heirarchical Metric Learning

Nicolas Beltran
Department of Computer Science
Columbia University
New York City, NY 10027
nb2838@columbia.edu

Ketan Jog
Department of Computer Science
Columbia University
New York City, NY 10027
kj2473@columbia.edu

Abstract

Many problems require a well defined notion of a distance between points in space. Constructing or finding such a measure falls into the field of metric learning. Although many algorithms exist in the field when a learner has access to a fixed dataset, there is room for improvement in terms of samples efficiency that the learner needs to know, imposition of desired structure, especially when the data appears in an *online* manner. We propose a project that reduces the problem of online/active metric learning to bandits. In case our plan turn out to be too ambitious, we have a fallback - an empirical investigation of some algorithms that have dealt with the problem in an online setting or in situations where the learner can selective query the points that it wants to know information about.

1 Introduction

2 Related Work

3 Long-term goals

4 Preliminaries

5 Problem Statement

We consider two different problems. The first problem consists of making a series of sequential predictions while learning a similiarity measure. We refer to this problem as online similarity prediction. The second problem consists of learning a similarity measure while querying points in space. We refer to this problem as active similarity learning. A precise description of both problems is provided below.

5.1 Online similarity learning

We consider an online similarity learning problem played over T rounds. At round t the environment samples K pairs of points $(\mathbf{x}^1_{t,k}, \mathbf{x}^2_{t,k}) \in \mathbb{R}^{2n}$. The agent then chooses pair $k_t \in [K]$ and is given a reward $r_{t,k_t} \in \{1, -1\}$. We assume that there exists some similarity function unknown to the agent $\phi : \mathbb{R}^{2n} \to \{-1, 1\}$ and that the rewards are such that if at time $t \in [T]$ the agent chooses pair $(\mathbf{x}^1_{t,k}, \mathbf{x}^2_{t,k})$ then the reward is $\phi(\mathbf{x}^1_{t,k}, \mathbf{x}^2_{t,k})$.

As usual we define the regret as

$$R_T = \mathbb{E}\left[\sum_{t=1}^{T} \phi(\mathbf{x}^1_{t,k^\star_t}, \mathbf{x}^2_{t,k^\star_t}) - \phi(\mathbf{x}^1_{t,k_t}, \mathbf{x}^2_{t,k_t})\right]$$

where $k^\star_t = \text{argmax}_{k\in[K]}\phi(\mathbf{x}^1_{t,k}, \mathbf{x}^2_{t,k})$

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

5.2 Active similarity learning

We assume that the learner has access to a dataset $D = \{\mathbf{x}_i \in \mathbb{R}^n | i \in [N]\}$ of unlabeled points and that there exists some function $\phi : \mathbb{R}^{2n} \to \{-1, 1\}$ which the learner is trying to learn. The learner can query T pairs of points in this set D to obtain a dataset $D_T = \{(\mathbf{x}_t^1, \mathbf{x}_t^2, y_t) \mid t \in [T]\}$. We assume that the learner mantains and estimate $\hat{\phi}_t \in \mathcal{F}$ of ϕ, where $\mathcal{F}$ is its function class and denote the loss between an estimate $\hat{\phi}$ a ϕ as

$$\mathcal{L}(\phi, \hat{\phi}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}\times\mathcal{D}}[(\hat{\phi}(\mathbf{x}, \mathbf{y}) - \phi(\mathbf{x}, \mathbf{y}))^2]$$

It's objective is to find $\min_{\phi\in\mathcal{F}} \mathcal{L}(\hat{\phi}_T, \phi)$

6 Description of the algorithms

In total we provide 4 different algorithms which we describe below. However, in reality they can be thought of as 2 different algorithms (a neural and linear model for similarity) which contain slight modifications to accomodate for the online similarity learning problem and the active similarity learning problem. We refer to our algorithms as OnSim-LinUCB, ActSim-LinUCB,OnSim-NeuralUCB, ActSim-NeuralUCB, depeding on whether they are based on LinUCB or NeuralUCB and on whether they attempt to solve the active or online formulation of our problem. We provide the four versions below with their slightly different modifications for completeness.

6.1 Online similarity learning

For our problem of online similarity learning we adopt the frameworks of LinUCB as described in [1] and NeuralUCB as described in [4]. We explain this in detail below.

6.1.1 OnSim-LinUCB

Our first algorithm performs a straighforward reduction of the online similarity learning problem to that of regular contextual bandits. We do this by assuming a linear structure on the similarity.

Mathematically, if we adopt the formulation we proposed above (5.1), we choose to model the similarity of two points as $\phi(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{A}\mathbf{y}$. We can see that this is a reasonable thing to do if we consider that

$$\phi(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{A}\mathbf{y} = \sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{x}_i \mathbf{y}_j \mathbf{A}_{i,j}$$

which we is linear in $\mathbf{A}$ and thus allows us to use the framework of LinUCB for our problem. Attending to this fact, one can see that Algorithm 1 is almost identical to the first algorithms in [1].

Algorithm 1: OnSim-LinUCB

Input : Rounds T and exploration parameter α

$A \leftarrow I_{n^2}$;
$b \leftarrow 0_{n^2}$;
for $t \in [T]$ **do**
  $\theta_t \leftarrow A^{-1}b$ Observe K pairs of vectors $x^k \in \mathbb{R}^n$, $y^k \in \mathbb{R}^n$;
  Create $z_{t,k} = (x_1^k y_1^k, x_1^k y_1^k, \ldots, x_n^k y_{n-1}^k, x_n^k y_n^k)$;
  for $k \in [K]$ **do**
    $p_{t,a} \leftarrow \theta_t^\top z_{t,k} + \alpha\sqrt{z_{t,k} A^{-1} x_{t,a}}$
  end
  Choose action $a_t = \text{argmax}_a p_{t,a}$ with ties broken arbitrarily;
  Observe payoff $r_t \in \{-1, 1\}$;
  $A \leftarrow A + z_{t,a_t} z_{t,a_t}^\top$;
  $b \leftarrow z_{t,a_t} r_t$;
end

6.1.2 OnSim-NeuralUCB

Unsurprisingly, the expressive power of the above framework is quite limited. The limitation comes from the fact that Algorithm 1 essentially assumes that similarity is the result of a dot prouct in which both points are independently mapped to a new representation via a linear transformation. This will obviously not be true for many datasets. To overcome these limitations we adapt NeuralUCB [4] to our circumstances.

Formally, we assume that

$$\phi(x, y) = \cos\left(f(x;\theta), f(y;\theta)\right) = \frac{\langle f(x;\theta), f(y;\theta)\rangle}{||f(x;\theta)||_2||f(y;\theta)||_2}$$

Here $f : \mathbb{R}^n \to \mathbb{R}^m$ is a simple neural network of the form

$$f(x;\theta) = b_L + W_L\sigma\left(b_{L-1} + W_{L-1}\sigma\left(\ldots\sigma\left(b_1 + W_1 x\right)\right)\right)$$

for some $L \in \mathbb{N}_{\geq 2}$, $W_1 \in \mathbb{R}^{d\times n}$, $W_L \in \mathbb{R}^{m\times d}$, $W_l \in \mathbb{R}^{d\times d}$ if $l \in [2, L-1]$, $b_L \in \mathbb{R}^m$, $b_l \in \mathbb{R}^d$ for $l \neq L$, and where $\sigma(x) = \max\{0, x\}$ applied component wise. We use θ to denote the flattened vector of $(W_L, b_L, \ldots, W_1, b_1)$. Using this notation OnSim-NeuralUCB is described below in Algorithm 2. We use p to denote the total number of trainable parameters in the neural network.

Algorithm 2: OnSim-NeuralUCB

Input : Rounds T, exploration parameter α, τ_r frequency of resets, τ_T frequency of training, E epochs for training, b_s batch size for training, ϵ learning rate.

$A \leftarrow I_p$;
for $t \in [T]$ **do**
  Observe K pairs of vectors $x_t^k \in \mathbb{R}^n$, $y_t^k \in \mathbb{R}^{\kappa}$;
  for $k \in [K]$ **do**
    $p_{t,k} \leftarrow \phi(x, y) + \alpha\sqrt{(\nabla_\theta\phi(x_t^k, y_t^k))^\top A^{-1}\nabla_\theta\phi(x_t^k, y_t^k)}$;
  end
  Choose pair $k_t = \text{argmax}_k p_{t,k}$ with ties broken arbitrarily;
  Observe payoff $r_t \in \{-1, 1\}$;
  if $t \mod \tau_r = 0$ **then**
    $\theta \leftarrow \text{Train}(\epsilon, E, \{(x_i^{k_i}, y_i^{k_i}, r_i)\}_{i=1}^t, b_s)$;
  end
  if $t \mod \tau_T = 0$ **then**
    $A \leftarrow I_p$
  end
  $A \leftarrow A + \nabla_\theta\phi(x_t^k, y_t^k)(\nabla_\theta\phi(x_t^k, y_t^k))^\top$;
end

Intuitively, this algorithm acts optimally with the current set of parameters that it is given and every so often it stops back to train the network. In Algorithm 3 we simply describe the normal process for training a classifier with mean squared error loss and assume that a normal gradient based algorithm is used. In the description we use Stochastic Gradient Descent as done in [4] but we believe that despite lack of theoretical justification, more modern optimization strategies such as Adam [3] do a better job in practice.

Although our algorithm is very similar to that proposed in [4] it differs in some key aspects. First, and perhaps most importantly, we adopt a very different architecture from that proposed in the original paper. In it, the authors assume that the layers of the neural network have no bias, assume a particular initialization scheme, don't use a cosine similarity function at the end of the neural network, nor do they implement a siamese network as we do. Second, we restart A throughout training. Third, we use a constant exploration parameter throughout the duration of the algorithm.

We propose these changes because we believe they are more sensible for our particular application (and do provide better empirical performance) but they destroy the theoretical guarantees provided by

original paper in which they were used, alongside arguments to the neural tangent kernel matrix [2] to provide a $\tilde{O}(\sqrt{T})$, bound on the regret of algorithm.

Algorithm 3: Train

Input : ϵ learning rate, E number of epochs, dataset $\{(x_i^{k_i}, y_i^{k_i}, r_i)\}_{i=1}^t$, b_s batch size

optimizer $\leftarrow$ SGD(ϵ, θ);
for $j = 1, \ldots, E$ **do**
  $\mathcal{D} \leftarrow \{(x_1^{k_1}, y_1^{k_1}, r_1), \ldots, (x_t^{k_t}, y_t^{k_t}, r_t)$;
  while $\mathcal{D}$ *is not empty* **do**
    Sample minibatch M of size b_s from the training examples;
    $l = \frac{1}{M} \sum_{(x_i^{k_i}, y_i^{k_i}, r_i) \in M} (\phi(x_i^{k_i}, y_i^{k_i}) - r_i)^2$;
    $\theta \leftarrow$ update using optimizer with loss l;
    Remove M from $\mathcal{D}$;
  end
end

6.2 Active Similarity Learning

Despite the differences with online similarity learning, for the active version of the problem we modify the above algorithms only slightly. The idea is to use the optimism bonus as a measure of uncertainty and ignore completely the reward. Using this framework, each time we have to make a query we can simply sample uniformly a subset of points in the unlabeled dataset $\mathcal{D}$ and then we reveal the label of the pair with the highest uncertainty as measured by the bonus. Although we use the bandits notation that we were using above, it is important to emphasize that r no longer represents a reward but rather the labels for the similarity of the points themselves. Algorithms 4 and 5 describe these processes formally. In Algorithm 4 the most important change is line 5 and in algorithm 5 the most important change is line 8.

Algorithm 4: Active-NeuralUCB

Input : Queries T, exploration parameter α, τ_r frequency of resets, τ_T frequency of training, E epochs for training, b_s batch size for training, ϵ learning rate.

$A \leftarrow I_p$;
for $t \in [T]$ **do**
  Sample K pairs of vectors $x_t^k \in \mathbb{R}^n$, $y_t^k \in \mathbb{R}^n$ from dataset D;
  for $k \in [K]$ **do**
    $p_{t,k} \leftarrow \alpha \sqrt{(\nabla_\theta \phi(x_t^k, y_t^k))^\top A^{-1} \nabla_\theta \phi(x_t^k, y_t^k)}$;
  end
  Choose pair $k_t = \text{argmax}_k p_{t,k}$ with ties broken arbitrarily;
  Query label $r_t \in \{-1, 1\}$;
  if $t \mod \tau_r = 0$ **then**
    $\theta \leftarrow$ Train($\epsilon, E, \{(x_i^{k_i}, y_i^{k_i}, r_i)\}_{i=1}^t, b_s$);
  end
  if $t \mod \tau_T = 0$ **then**
    $A \leftarrow I_p$
  end
  $A \leftarrow A + \nabla_\theta \phi(x_t^k, y_t^k)(\nabla_\theta \phi(x_t^k, y_t^k))^\top$;
end

Algorithm 5: Active-LinUCB

Input : Rounds T and exploration parameter α

$A \leftarrow I_{n^2}$;
$b \leftarrow 0_{n^2}$;
for $t \in [T]$ **do**
  $\theta_t \leftarrow A^{-1}b$ Observe K pairs of vectors $x^k \in R^n$, $y^k \in R^n$;
  Create $z_{t,k} = (x_1^k y_1^k, x_1^k y_1^k, \ldots, x_n^k y_{n-1}^k, x_n^k y_n^k)$;
  for $k \in [K]$ **do**
    $p_{t,a} \leftarrow \alpha\sqrt{z_{t,k}A^{-1}x_{t,a}}$
  end
  Choose action pair $k_t = \text{argmax}_a p_{t,a}$ with ties broken arbitrarily.;
  Observe label $r_t \in \{-1, 1\}$;
  $A \leftarrow A + z_{t,a_t} z_{t,a_t}^\top$;
  $b \leftarrow z_{t,a_t} r_t$;
end

7 Experiments

8 Conclusion

References

[1] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 208–214, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[2] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. 2018.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[4] Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural contextual bandits with ucb-based exploration, 2019.