

Active and Online Similarity Learning

An application of bandits to Similarity learning

Ketan Jog and Nicolas Beltran

Columbia University

May 2021

What we did:

Use bandit algorithms to solve the following problems:

- Online learning of a similarity measure.
- Active learning of a similarity measure (i.e. by querying labels).

Definition

Similarity Measure: A function $\phi : \mathbb{R}^{2n} \rightarrow \mathbb{R}$ that maps datapoints $x, y \in \mathbb{R}^n$ to a real number in \mathbb{R} according to how “similar” they are.

Problem: Online Similarity Learning

At round t the environment samples K pairs of points $(x_{t,k}, y_{t,k}) \in \mathbb{R}^{2n}$. We choose pair $k_t \in [K]$ and get reward $r_{t,k_t} \in \{1, -1\}$ based on whether they are similar.

We are trying to minimize:

$$R_T = \mathbb{E} \left[\sum_{t=1}^T r_{t,k_t^*} - r_{t,k_t} \right]$$

where $k_t^* = \operatorname{argmax}_{k \in [K]} \phi(x_{t,k}, y_{t,k})$

Problem: Active Similarity Learning

Learner has access to a dataset $D = \{x_i \in \mathbb{R}^n | i \in [N]\}$ of unlabeled points. The learner can query T pairs of points in this set D to obtain a dataset $D_T = \{(x_t, y_t, r_t) \mid t \in [T]\}$. The learner maintains and estimate $\hat{\phi}_t \in \mathcal{F}$ of ϕ , where \mathcal{F} is its function class. Denote the loss between an estimate $\hat{\phi}$ a ϕ as

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

Goal is to find

$$\min_{\hat{\phi}_T \in \mathcal{F}} \mathcal{L}(\hat{\phi}_T, \phi)$$

First idea:

Assume that

$$\phi(x, y) = x^\top Ay = \sum_{i=1}^n \sum_{j=1}^n x_i y_j A_{i,j}$$

and then use linucb like methods.

Pros:

- Easy to reason about.
- Fast.

Cons:

- Can only learn linear similarities.

First idea Solution to problem 1:

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 $z_{t,k} = (x_1^k, y_1^k, x_1^k, y_1^k, \dots, x_n^k y_{n-1}^k, x_n^k, y_n^k);$

for $k \in [K]$ **do**

$p_{t,a} \leftarrow \theta_t^\top z_{t,k} + \alpha \sqrt{z_{t,k}^\top A^{-1} z_{t,k}}$

end

 Choose action $k_t = \operatorname{argmax}_a p_{t,a}$ with ties broken arbitrarily;

 Observe payoff $r_t \in \{-1, 1\};$

$A \leftarrow A + z_{t,k_t} z_{t,k_t}^\top;$

$b \leftarrow z_{t,k_t} r_t;$

end

Algorithm 1: OnSim-LinUCB

Second idea: Solution to active learning problem

Use the ideas about optimistic reward to encourage the learner to explore and choose points optimally.

We replace

$$p_{t,a} \leftarrow \theta_t^\top z_{t,k} + \alpha \sqrt{z_{t,k}^\top A^{-1} z_{t,a}}$$

by

$$p_{t,a} \leftarrow \sqrt{z_{t,k}^\top A^{-1} z_{t,a}}$$

Second Idea: Solution to active learning problem

Input: Rounds T and exploration parameter α

$A \leftarrow I_{n^2};$

$b \leftarrow 0_{n^2};$

for $t \in [T]$ **do**

$\theta_t \leftarrow A^{-1}b$ Sample K pairs of vectors $x^k \in R^n, y^k \in R^n;$

 Create $z_{t,k} = (x_1^k y_1^k, x_1^k y_1^k, \dots, x_n^k y_{n-1}^k, x_n^k y_n^k);$

for $k \in [K]$ **do**

$p_{t,a} \leftarrow \alpha \sqrt{z_{t,k} A^{-1} x_{t,a}}$

end

 Choose action pair $k_t = \operatorname{argmax}_a p_{t,a}$ with ties broken arbitrarily.;

 Observe label $r_t \in \{-1, 1\};$

$A \leftarrow A + z_{t,k_t} z_{t,k_t}^\top;$

$b \leftarrow z_{t,j_t} r_t;$

end

Algorithm 2: ActiveSim-LinUCB

More expressive model classes: Neural UCB

Method for solving the problem using a neural network to predict the rewards.

- Uses a Neural Network to model the rewards.
- Probably correct regret bound of $\tilde{O}(\tilde{d}\sqrt{T})$.
- Slow due to the computation of a very wide matrix.

Second model

Assume that

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

or

$$\phi(x, y) = \langle f(x; \theta), f(y; \theta) \rangle$$

where

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

Algorithm for solving Online Similarity Learning Problem

```
 $A \leftarrow I_p;$ 
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  $k_t = \operatorname{argmax}_k p_{t,k}$  with ties broken arbitrarily;
    Observe payoff  $r_t \in \{-1, 1\};$ 
    if  $t \bmod \tau_r = 0$  then
         $\theta \leftarrow \operatorname{Train}(\epsilon, E, \{(x_i^{k_i}, y_i^{k_i}, r_i)\}_{i=1}^t, b_s);$ 
    end
    if  $t \bmod \tau_T = 0$  then
         $A \leftarrow I_p$ 
    end
     $A \leftarrow A + \nabla_{\theta} \phi(x_t^{k_t}, y_t^{k_t})(\nabla_{\theta} \phi(x_t^{k_t}, y_t^{k_t}))^{\top};$ 
end
```

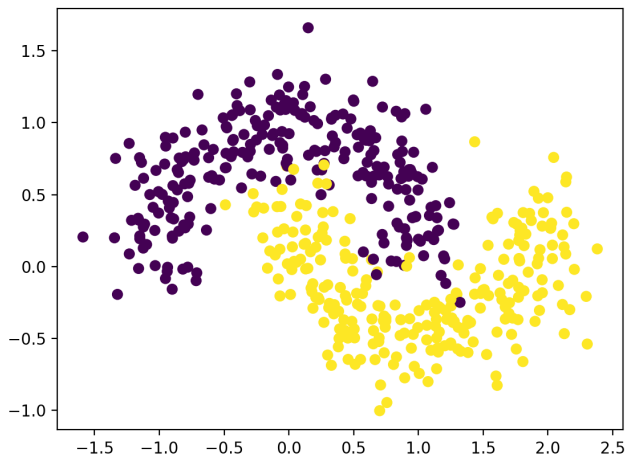
Algorithm 3: OnSim-NeuralUCB

Algorithm for solving Active Learning Problem

```
 $A \leftarrow I_p;$   
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  $k_t = \operatorname{argmax}_k p_{t,k}$  with ties broken arbitrarily;  
    Query label  $r_t \in \{-1, 1\};$   
    if  $t \bmod \tau_r = 0$  then  
         $\theta \leftarrow \operatorname{Train}(\epsilon, E, \{(x_i^{k_i}, y_i^{k_i}, r_i)\}_{i=1}^t, b_s);$   
    end  
    if  $t \bmod \tau_T = 0$  then  
         $A \leftarrow I_p$   
    end  
     $A \leftarrow A + \nabla_{\theta} \phi(x_t^{k_t}, y_t^{k_t})(\nabla_{\theta} \phi(x_t^{k_t}, y_t^{k_t}))^{\top};$   
end
```

Algorithm 4: ActiveSim-NeuralUCB

Empirical Test with Crescent Moons:



Testing a SVM on learned features

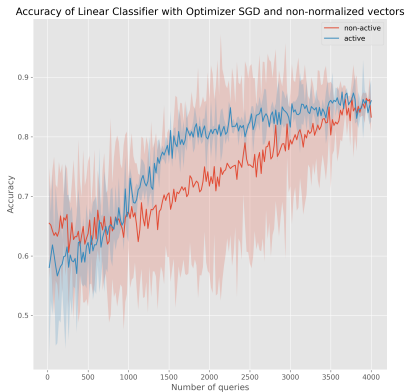
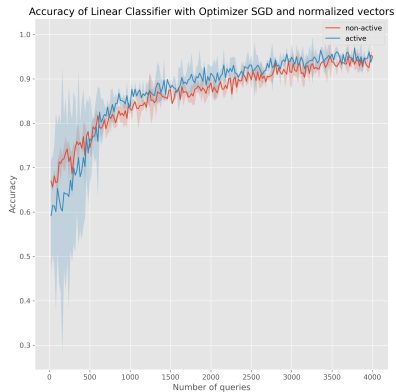


Figure: Performance SVM

Comparing L2 loss of learned features

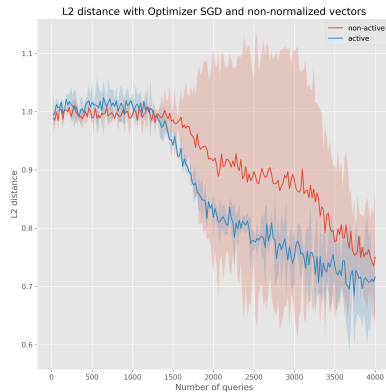
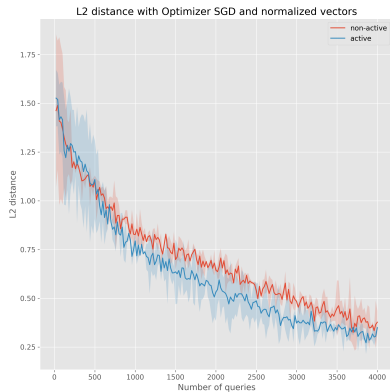


Figure: Performance SVM

Comparison of regret

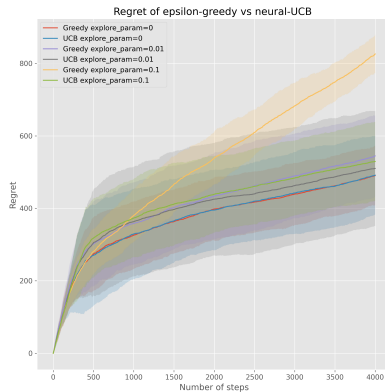
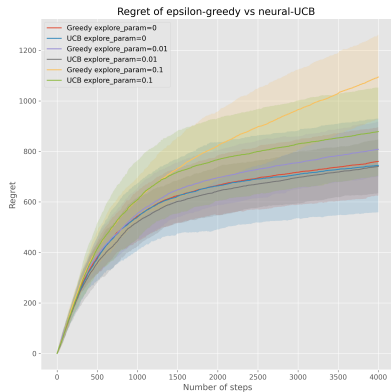


Figure: L2 Loss

Caveats:

- The algorithm is really slow (we tried using on raw MNIST/CIFAR10 etc and it was unusable).
- The results seem to be sensitive with respect to the optimizer (see next slide).
- We currently don't have theoretical guarantees on the regret.

Conclusion: Is this the right approach?

Maybe but not exactly like this.