Report for Data Science Project 2

Xiaoshuang Chen†, Xu Lin†, Xinpeng Liu†, Yue Xu†,

***Abstract*— Distance metric is a important subject in machine learning, especially to K nearest neighbors algorithm (KNN), since by choosing a proper distance metric on data set, some indivisible problems can be transformed into linear problems. In this paper, we evaluated the performance of some popular distance metrics on Animals with Attributes (AwA2) data set. During the experiments, we implemented KNN with CUDA to accelerate the inference, whose source code is made public. We also implemented and compared various popular metric learning algorithms and analyzed the difference of them. Furthermore, we did a comprehensive and thorough research on cosine distance and L_p normalization and gave our proofs, conclusions and hypothesis.**

I. INTRODUCTION

Distance metric is a important subject in machine learning since by choosing a proper distance metric on data set, some indivisible problems can be transformed into linear problems. And due to the low extra computational burden of different distance metrics, it becomes fundamental to select a metric that match the practical problem. The algorithm that most dependent to metric is K-nearest neighbors (KNN), for it makes prediction fully relying on the distance to the samples in training set. The choice of distance metric can significantly affects the performance of KNN as well as other similar algorithms.

Distance metrics have developed for decades. For **vector metric**, the simplest and most commonly used one is Minkowsky distance, namely L_p distance, which can be regarded as the L_p norm of the difference of two vectors, including Manhattan, Euclidean, Chebyshev distance and so on. Another simple metric is cosine distance, which is widely used in text processing. For **probability distribution metric**, EMD (Earth's Mover Distance), MMD (Maximum Mean Discrepancy) are most typical ones. Divergence is also popular for probability metric, including KL divergence, JD divergence and Bregman divergence, though most of them do not strictly satisfy the rules of metrics. Nowadays, **metric learning** methods grows rapidly since they exploit the information of data distribution or data labels. Metric learning tries to learn the optimal matrix M in Mahalanobis distance. Some of the learning algorithms are LMNN[1], contrastive loss[2], NCA[3], semi-hard mining[4] and binomial deviance loss.

In this paper, we evaluated the performance of some popular distance metrics on Animals with Attributes (AwA2) data set[5], including Minkowsky distance, cosine distance, Bregman divergence, LMNN[1], contrastive loss[2], NCA[3], semi-hard mining[4] and binomial deviance loss. We wrote `CUDA` implementation of KNN and compared the KNN classification accuracy with different distance metrics. The metrics and algorithms are introduced in detail and experimented in section II and section III. Moreover, we met some problems or raised some conjectures from our results, and conducted further experiments to explain the reasons of our results in section IV.

There are three main highlights of our work. First, we used `CUDA` and GPU to implement KNN algorithm and significantly accelerated the prediction. We also made the source code public[1]. Second, we implemented and compared various popular metric learning algorithms and analyzed the difference of them. Third, we did a comprehensive and thorough research on cosine distance and L_p normalization. We gave our proofs, conclusions and hypothesis in subsection IV-A.

II. METHOD

A. KNN and CUDA Implementation

KNN (K nearest neighbors) is a simple and non-parameter algorithm widely used in supervised learning. For each test sample, KNN gives a prediction based on the nearest K samples in the training set. Specifically, for classification problems, the sample is classified by a plurality vote of its K nearest neighbors. As KNN relies on the distance of samples to classify, the distance metric is decisive to its performance. We will discuss the impact of distance metrics on KNN in the most part of this paper.

KNN also suffers from high computational complexity at inference because it computes the distance between every test sample and training sample. Thanks to the rapid development of parallel computing chips like GPUs, we implemented KNN with different distance metrics with CUDA on GPU, which significantly accelerate the inference of the algorithm. The KNN classification on CUDA can be divided into 3 steps:

1) Compute the distance between N_{tr} training samples and N_{te} test sample;
2) (Partially) sort distance vector of each test samples;
3) Choose the nearest K neighbors of each test samples and give the plurality vote result.

In the second step, our choice of sort algorithm is heap sort because recursive quick sort and merge sort which has high space complexity are not suitable for CUDA. And heap sort can also be modified to partial sort algorithm to reduce the unnecessary computation. In step 3, we can calculate the accuracy of different K's at one time since the plurality

† Equal contribution.

[1] https://github.com/silicx/KNN-PyCUDA

vote can be progressively computed. The time complexity of three steps are $O(N_{tr}N_{te}D)$, $O(N_{tr} + K_m \log N_{tr})$ and $O(N_{te}K_m)$ respectively, where D is sample dimensionality and K_m is the number of K in KNN that we want to evaluate. Our implementation detail and results are shown in subsection III-B.

B. Simple Distance Metrics

1) Minkowsky distance: Minkowsky distance is the most popular distance metric due to its simplicity. It can be seen as the L_p norm of the difference between two vectors:

$$dist_p(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_p = \left(\sum_{i=1}^{d}(x_i - y_i)^p\right)^{\frac{1}{p}} \quad (1)$$

The most commonly used p is 1, 2 and ∞, respectively named Manhattan, Euclidean and Chebyshev distance:

$$\begin{aligned} dist_{\text{manh}}(\boldsymbol{x}, \boldsymbol{y}) &= \|\boldsymbol{x} - \boldsymbol{y}\|_1 = \sum_{i=1}^{d}|x_i - y_i| \\ dist_{\text{eucl}}(\boldsymbol{x}, \boldsymbol{y}) &= \|\boldsymbol{x} - \boldsymbol{y}\|_2 = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^T(\boldsymbol{x} - \boldsymbol{y})} \\ dist_{\text{cheb}}(\boldsymbol{x}, \boldsymbol{y}) &= \|\boldsymbol{x} - \boldsymbol{y}\|_\infty = \max|x_i - y_i| \end{aligned} \quad (2)$$

2) Cosine similarity and its variants: Cosine similarity measures the similarity between two vectors by the cosine of their angle. Given two vectors $\boldsymbol{x}, \boldsymbol{y}$, the cosine similarity of them is defined as:

$$\begin{aligned} Sim_{cos}(\boldsymbol{x}, \boldsymbol{y}) &= \cos < \boldsymbol{x}, \boldsymbol{y} > \\ &= \frac{\boldsymbol{x}^T\boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} \end{aligned} \quad (3)$$

which ranges from -1 to 1. Since cosine similarity measures the similarity of vectors rather than distance, we define **cosine distance** to evaluate the distance of two samples:

$$dist_{cos} = \frac{1}{2}(1 - Sim_{cos}) \quad (4)$$

Note that cosine distance is only a pre-metric instead of metric which varies from 0 to 1, but it still can be used as a distance metric in many algorithms.

A popular variant of cosine similarity is ADJUSTED COSINE SIMILARITY which is done by subtract the mean of samples. This resembles the cosine similarity of z-score normalized data and we implemented and studied normalized cosine distance in section III instead of adjusted one.

3) Divergences: Bregman divergence[6] is a measurement of distance between two points or probability distributions. One of the cases of Bregman divergence is squared Euclidean distance.

Let $F : \Omega \to \mathbb{R}$ be a continuously-differentiable, strictly convex function defined on a closed convex set Ω. The Bregman divergence associated with F for points $p, q \in \Omega$ is the difference between the value of F at point p and the value of the first-order Taylor expansion of F around point q evaluated at point p:

$$D_F(p, q) = F(p) - F(q) - \langle \nabla F(q), p - q \rangle \quad (5)$$

Bregman divergence has four properties: Non-negativity, Convexity, Linearity and Duality.

Different convex function will generate different cases of Bregman divergences. Here lists some examples of distance/divergence(Squared Euclidean distance, generalized KL divergence and ItakuraSaito distance) and the corresponding convex function F:

$F(x)$	$D_F(x, y)$
$\|x\|^2$	$\|x - y\|^2$
$\sum_i p(i) \log p(i)$	$\sum_i p(i) \log \frac{p(i)}{q(i)} - \sum p(i) + \sum q(i)$
$-\sum_i \log p(i)$	$\sum_i (\frac{p(i)}{q(i)} - \log \frac{p(i)}{q(i)} - 1)$

TABLE I
BREGMAN DIVERGENCES

C. Metric Learning

Simple distance metrics mentioned in subsection II-B can usually provide passable results for algorithms like KNN. However, since simple standard distance metrics could ignore some important properties for specific tasks, the results may be non-optimal.

Here comes the idea of metric learning: project samples to another space, where the simple distance metrics could work better for specific tasks. If we limit the simple distance metric to Euclidean distance, what we want to learn is actually a Mahalanobis distance[7] as Equation 6 shows:

$$\mathrm{d}(x_1, x_2) = (XL)^T(XL) = X^T L^T L X \quad (6)$$

The goal is to learn a suitable L for a specific task. Many methods of metric learning have been proposed, like LMNN, NCA, IMTL, etc. Also, some dimensionality reduction methods could also be seen as metric learning, such as LDA.

In this paper, we performed 6 popular metric learning methods, including neighborhood based methods like LMNN and information theory based methods like NCA and MCML.

1) Large Margin Nearest Neighbor: LMNN[1] is a supervised metric learning method that tries to learn a distance metric for KNN classifier. The main intuition behind LMNN is to learn a pseudo-metric under which all data instances in the training set are surrounded by at least k instances that share the same class label. Thus, the leave-one-out error of KNN can be minimized.

Let $\{(x_i, y_i)\}_{i=1}^n$ denote a training set of n labeled examples with $x_i \in \mathcal{R}^d$ and class labels y_i. y_{ij} denotes x_i and x_j belong to the same class. Our goal is to learn a distance metric $d(x, y)$, which is computed as Equation 7:

$$d(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^T L^T L(\boldsymbol{x} - \boldsymbol{y}) \quad (7)$$

, where L is initialized to unit matrix of size $d \times d$. We first use $d(x, y)$ to compute distance matrix $D_{n\times n}$. Then we define η_{ij}, which indicates whether x_j is one of the k-nearest neighbors that share the same label with x_i. If $\eta_{ij} = 1$, we call x_j is a target neighbor of x_i. $\eta_{n\times n}$ won't be changed during the whole learning procedure. Then we minimize the object function described in Equation 8.

$$\begin{aligned} J(L) &= \sum_{ij} \eta_{ij} d(x_i, x_j) + c \sum_{ijl} \eta_{ij}(1 - y_{il}) z_{ijl} \\ z_{ijl} &= max(1 + d(x_i, x_j) - d(x_i, x_l), 0) \end{aligned} \tag{8}$$

$J(L)$ is composed of two terms. The first term penalizes large distances between each sample and its target neighbors. The second term penalizes the small distances between each sample and all the other samples except its target neighbors. By optimizing this, the goal can be achieved.

2) Contrastive Loss: Contrastive loss[2] was first proposed for dimensionality reduction. However, it can also be used for metric learning without many changes. The learning solely relies on the neighborhood relations and doesn't need any distance measure in the input space. The main idea of Contrastive loss is to find a mapping, which maps similar samples to close points and dissimilar samples to distant points.

Let $X_{n\times d} = \{x_1; x_2; \ldots; x_n\}$ denote the original samples, $G_M(x_i) = Mx_i$ denote the mapped points of x_i, and $D_M(x_i, x_j)$ denote the defined distance between $G_M(x_i)$ and $G_M(x_j)$. And define $y_{ij} = 1$ if x_i and x_j are dissimilar, otherwise we set $y_{ij} = 0$. The objective function for a given transform matrix M can be defined as

$$\begin{aligned} L(M, x_i, x_j) =& (1 - y_{ij}) D_M(x_i, x_j) + \\ & y_{ij} max(c - D_M(x_i, x_j), 0) \end{aligned} \tag{9}$$

, where c is a margin. The function is quite ocular, punishing the distances between similar points and ensure dissimilar points' distances bigger than a given value.

3) Neighbourhood Components Analysis: NCA[3] is a supervised metric learning method that aims to maximize the expected number of points correctly classified based on softmax normalization.

In particular, each point i selects another point j as its neighbor with some probability p_{ij}, and inherits its class label from the point it selects. The probability p_{ij} is defined using a softmax over Euclidean distances in the transformed space:

$$p_{ij} = \frac{\exp\left(-\|Lx_i - Lx_j\|^2\right)}{\sum_{k\neq i} \exp\left(-\|Lx_i - Lx_k\|^2\right)} \quad , \quad p_{ii} = 0 \tag{10}$$

where L is the transformation matrix.

Under this stochastic selection rule, we can compute the probability p_i that point i will be correctly classified (denote the set of points in the same class as i by $C_i = \{j | c_i = c_j\}$):

$$p_i = \sum_{j\in C_i} p_{ij} \tag{11}$$

Thus, the objective the model aims to maximize is the expected number of points correctly classified under this scheme:

$$f(A) = \sum_i \sum_{j\in C_i} p_{ij} = \sum_i p_i \tag{12}$$

4) Semi-Hard Mining Strategy: Semi-Hard Mining Strategy[4] is a method with margin strategy for metric learning. As a supervised metric learning method, it aims to ensure a point x_i is closer to all other points x_i^p (positive) of the same label than it is to any point x_i^n (negative) of different labels. The points x_i, x_i^p, x_i^n can be called a triplet.

Thus we want:

$$\begin{aligned} &\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2, \\ &\forall (f(x_i^a), f(x_i^p), f(x_i^n)) \in \mathcal{T}. \end{aligned} \tag{13}$$

where α is a margin that is enforced between positive and negative pairs. $\mathcal{T}$ is the set of all possible triplets in the training set.

The loss to be minimized is then

$$L = \sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]_+ . \tag{14}$$

5) Binomial Deviance Loss: This model uses binomial deviance as the cost function, which is formulated as

$$J_{dev} = \sum_{i,j} W \circ \ln\left(e^{-\alpha(S-\beta)\circ M} + 1\right), \tag{15}$$

where $\circ$ is element-wise matrix product, and M, W is defined as

$$\begin{aligned} M &= [M_{ij}]_{n\times n}, \quad M_{ij} = \begin{cases} 1, & \text{positive pair} \\ -1, & \text{negative pair} \\ 0, & \text{neglected pair} \end{cases} \\ W &= [W_{ij}]_{n\times n}, \quad W_{ij} = \begin{cases} \frac{1}{n_1}, & \text{positive pair} \\ \frac{1}{n_2}, & \text{negative pair} \\ 0, & \text{neglected pair} \end{cases} \end{aligned} \tag{16}$$

and S is the similarity matrix. In our experiments, S was defined based on Euclidean distance:

$$S_{euc}(\mathbf{x}, \mathbf{y}) = -\sum_i (\mathbf{x}_i - \mathbf{y}_i)^2 \tag{17}$$

and α, β is hyper-parameters in this model, which is specified in the experiment section.

6) Maximally Collapsing Metric Learning: MCML[8] is a supervised distance metric learning technique, based on the idea that if all the samples of the same class were projected to the same point, and data of different classes were projected to different points and sufficiently far away, we would have an ideal class separation. The target is to learn a projection matrix L that satisfies this. To achieve this, it exploits the tools provided by the information theory. First, it introduces a probability distribution like that proposed in subsubsection II-C.3:

$$p^M(j|i) = \frac{\exp\left(-\|Lx_i - Lx_j\|^2\right)}{\sum_{k\neq i} \exp\left(-\|Lx_i - Lx_k\|^2\right)} \tag{18}$$

, where $p^M j|i$ indicates the probability that x_j will be classified with the class of x_i. And the ideal distribution we are looking for is a binary distribution for which the probability that a sample is correctly classified is 1, and 0 otherwise, that is,

$$p^0(j|i) = 1(y_i = y_j) \tag{19}$$

Thus the objective function can be defined as

$$\mathcal{J}(L) = \sum_{i=0}^{n} KL(p^0(*|i)|p^M(*|i)) \tag{20}$$

III. Experiment

In this section, we used various popular dimensionality reduction methods described in section II. If not mentioned, the algorithms were implemented with `Python 3`. Besides, `PyCUDA`[9] was used to improve the speed of KNN and `PyTorch` was used to accelerate metric learning algorithms.

A. Data Set

We evaluated the chosen dimensionality reduction method on Animals with Attributes (AwA2) data set[5]. This data set consists pre-extracted 2048-dimensional deep learning features for 37322 images of 50 animal classes. We split the images in each category into 60% for training and 40% for testing.

B. CUDA Implementation of KNN

We implemented KNN with different distance metrics with `PyCUDA`[9], which is a Python interface and JIT compiler of C++ CUDA kernel functions. As described in subsection II-A, the inference procedure of our approach has three steps: distance measurement, sorting and voting. To fully exploit the parellelism of GPU, our implementation concurrently computes the prediction of every samples on multiple parameter K (up to the number of training set) of KNN. All the source code are made public as mentioned before.

In experiments, we ran our implementation on Nvidia Tesla T4 with CUDA 10.0 and the metric for KNN is Euclidean distance. To compare the inference time of CPU and GPU, we tested various algorithms and implementations on the AwA2 dataset with all $K \in (0, 5000]$. The results are shown in Table II.

Implementation	Measure	Sort	Vote	Total
CPU(`sklearn`, brute)	/	/	/	42.6s*
CPU(`sklearn`, ball-tree)	/	/	/	1189.5s*
CPU(`sklearn`, Kd-tree)	/	/	/	1466.4s*
CPU(`Numpy`, brute)	3661.8s**	46.1s	240.9s**	3948.8**
GPU(`PyCUDA`, Tesla T4)	**24.50s**	**5.31s**	**0.80s**	**30.61s**

* Only for a single K.
** Estimated.

TABLE II
Inference Time of Different Implementations of KNN

All the methods returned correct results. It is evident that GPU significantly accelerated the computation of KNN. Traditional CPU implementations spent $40\times$ to $130\times$ time of GPU. However, the brute search method in Python `sklearn` package also have surprisingly fast performance, only $2\times$ of GPU. This may be due to the C++ optimization of `sklearn` kernel. So we are going to convert our `PyCUDA` implementation to a fully C++ CUDA version in the future to avoid Python code and further reduce the computing time.

C. Baseline: Simple Distance Metrics

We ran KNN on various simple distance metrics, including Minkowski and cosine distance. For each case, we tried hyper-perameter K from 1 to 2000 to select the best model and a small part of their classification accuracy is shown in Table III. The performance-K curves are shown in Figure 1.

K \ Metric	L1	L2	L_∞	cos
1	84.95%	85.62%	72.24%	86.69%
2	84.09%	84.73%	71.07%	85.60%
3	86.62%	87.46%	73.99%	88.75%
4	86.23%	87.36%	75.26%	88.92%
5	86.88%	88.08%	76.23%	89.43%
6	86.75%	88.00%	76.20%	89.38%
7	**87.18%**	88.37%	76.80%	89.64%
8	86.86%	88.00%	**76.92%**	89.64%
9	86.99%	88.31%	76.66%	89.78%
10	86.86%	88.32%	76.59%	89.79%
11	87.03%	88.43%	76.59%	**89.93%**
12	86.70%	**88.46%**	76.56%	89.83%
13	86.85%	88.27%	76.49%	89.90%
14	86.68%	88.05%	76.56%	89.89%
15	86.62%	87.94%	76.46%	89.89%
50	83.60%	85.99%	73.94%	88.51%
100	81.24%	83.86%	71.15%	87.02%
200	77.75%	80.98%	68.06%	85.22%
500	70.67%	76.14%	62.12%	81.84%
1000	62.65%	71.43%	56.37%	78.59%
Best	87.18%	88.46%	76.92%	**89.93%**

TABLE III
Result of Simple Distance Metrics

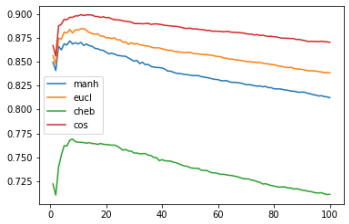


Fig. 1. Accuracy curve of simple distance metrics

All the performance curves are increasing first then decreasing and the best choices of K are mostly in the range of 5 to 15. The worst metric is L_∞ distance because it only uses one dimension that has maximal value to represent the whole vector. Unexpectedly, the best metric is cosine distance although it seemingly omits the length information of samples. We will comprehensively and thoroughly discuss the advantages of cosine metric in subsection IV-A.

Besides, we also tried to z-score normalize the original data to alleviate the heterogeneity of dimensions. Part of the

KNN classification performance after z-score normalization is shown in Table IV and Figure 2.

K \ Metric	L1	L2	L_∞	cos
1	83.50%	83.92%	61.53%	85.42%
2	82.20%	83.01%	60.53%	83.97%
3	84.98%	85.57%	63.68%	87.25%
4	84.76%	85.75%	65.16%	87.48%
5	85.26%	86.53%	66.17%	88.33%
6	85.10%	86.32%	66.37%	88.22%
7	**85.52%**	86.60%	66.50%	88.63%
8	85.22%	86.48%	66.84%	88.69%
9	85.38%	**86.64%**	67.22%	88.90%
10	85.02%	86.39%	67.17%	88.79%
11	85.04%	86.52%	**67.44%**	88.93%
12	84.82%	86.18%	67.21%	88.93%
13	84.93%	86.25%	67.10%	88.93%
14	84.72%	86.21%	67.24%	88.83%
15	84.62%	86.06%	67.35%	88.91%
16	84.55%	86.00%	67.36%	88.84%
17	84.36%	86.05%	67.30%	**88.93%**
50	81.36%	83.61%	64.61%	87.95%
100	78.42%	80.96%	61.40%	86.60%
200	73.50%	77.31%	57.71%	84.49%
500	64.33%	70.00%	51.08%	81.06%
1000	53.77%	62.36%	44.01%	77.65%
Best	85.52%	86.64%	67.44%	88.93%
Baseline	**87.18%**	**88.46%**	**76.92%**	**89.93%**

TABLE IV

RESULT OF SIMPLE DISTANCE METRICS WITH Z-SCORE NORMALIZATION

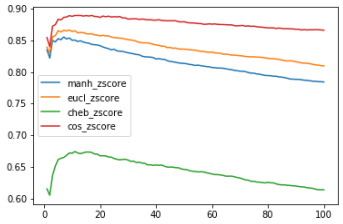


Fig. 2. Accuracy curve of simple distance metrics with z-score normalization

Z-score normalization didn't achieve higher accuracy as expected. Instead, the accuracy dropped 1% or more after normalization. The negative effect of z-score normalization may be because AwA2 dataset is pre-extracted deep learning features so its dimensions are not heterogeneous and z-score normalization omits some important information.

D. Bregman Divergence

Unfortunately, the results of two variants of Bregman Divergence, generalized KL Divergence and Itakura-Saito distance, were very bad. In the KNN algorithm, regardless of the value of K, the accuracy is a nearly 1.96%, which is similar to random classifier. A possible reason is that the features of AwA2 set do not have probabilistic semantics and are not suitable for probability metric like divergence as the distance metric in KNN algorithm.

E. Metric Learning

There are a lot of off-the-shelf codes for the popular metric learning algorithms. However, most of them are based on the whole training set and use only CPU, suffering from limited memory and speed. Therefore, we implemented these algorithms by ourselves using PyTorch, which helped us in accelerating the calculation. And to solve the memory problem, we introduced mini-batch strategy to the implementation. All the algorithm were optimized with RMSProp with learning rate of 1e-4. We trained each model for 10 epochs on the training set with batch size of 500. All the algorithms are evaluated on the testing set with KNN, using different ks. Notice that all the algorithms were only used to learning a Mahalanobis distance metric, though they could be used to learn other metrics like cosine similarity. Since some methods involves k in the training phase, to distinguish, we use k to denote k in the training procedure, and K for that in the testing phase.

1) LMNN: We trained our LMNN model with different ks, and the evaluation results were best when $k = 4$. The results are shown in Table V.

We can see it obviously improved the classification accuracy. Another interesting observation was that in this case the best performance was not achieved when $k = K$, though LMMN is aimed at optimizing the leave-one-out cross validation accuracy for k-nearest neighbor classifier. The reason is simple. Since enhancing the k nearest target neighborhood relationship is also enhancing all the target neighborhood relationship, LMNN with k could enhance KNN classifiers' performance regardless of K. To further illustrate the influence of k, we also trained our model with different ks. The results and observations will be shown in subsection IV-B.

2) Contrastive Loss: In implementation, we didn't use a fixed margin. Instead, we used a strategy similar to subsubsection II-C.4: we used the biggest distance between all the similar pairs as margin. The results are shown in Table V. As presented, the performance outperformed baseline obviously, and the best performance was achieved approximately at the similar k to baseline.

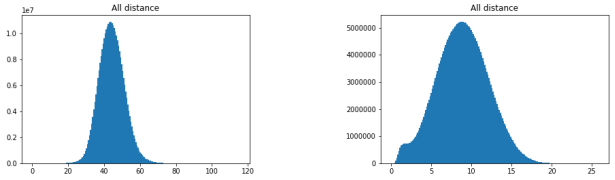
3) NCA: NCA aims to maximize $f(A)$, which is the expected number of points that correctly classified. Thus we defined the loss as $-f(A)$ for minimization. Due to the original data scale could result in overflow, we first performed L2 normalization on the data set before training and testing. The results of different k in KNN is shown in Table V.

4) Semi-Hard Mining Strategy: The results of different k in KNN are shown in Table V. In our experiments, we observed that even when the margin α was set to 0, the model still had considerable performance.

5) Binomial Deviance: In our experiment, we did not use a fixed hyper-parameter β. Instead, we used the biggest distance among positive pairs as β for negative pairs, and used the smallest distance among negative pairs as β for positive pairs. Thus, positive pairs which had smaller distance than any negative pairs would be neglected. And α was set to 2 in

our implement. The results of different k in KNN are shown in Table V.

6) Maximally Collapsing Metric Learning: Since the computation procedure involved exponentiation, the original distance scale could result in overflow. Therefore, the experiments were performed after L2 normalization. The results are shown in Table V. Besides the accuracy, we also plot the distance distribution before and after the MCML in Figure 3. Remind that the intuition of MCML is to project similar samples to the same points and dissimilar samples to distant points. From the small spike close to 0 in subsubsection III-E.6, we can tell the algorithm achieved to approach the goal.



(a) Before MCML (b) After MCML

Fig. 3. Distribution of Euclidean distance before and after MCML

7) Comparison of metric learning methods: We list the results of all the metric learning methods we implemented in Table V.

A \ M / K	LMNN	CL	NCA	SHS	BD	MCML
1	88.51%	88.54%	89.13%	90.56%	89.62%	88.31%
2	87.86%	88.01%	88.24%	89.56%	88.42%	87.80%
3	90.39%	89.96%	90.15%	91.45%	90.82%	89.99%
4	90.49%	90.37%	90.19%	91.47%	90.76%	90.41%
5	91.02%	90.80%	90.74%	92.10%	91.16%	90.82%
6	90.86%	90.74%	90.80%	91.83%	91.31%	91.05%
7	91.30%	90.97%	91.18%	92.02%	91.56%	91.24%
8	91.24%	91.09%	91.09%	92.01%	91.35%	91.20%
9	91.37%	91.14%	91.18%	91.96%	91.47%	91.25%
10	91.47%	91.08%	91.10%	92.13%	91.27%	91.30%
11	91.49%	91.06%	91.25%	92.01%	91.27%	91.35%
12	91.52%	91.10%	91.08%	92.04%	91.21%	91.35%
13	91.49%	91.13%	91.24%	92.03%	91.21%	91.36%
14	91.41%	91.02%	91.18%	92.07%	91.13%	91.37%
15	91.32%	91.02%	91.27%	92.07%	91.23%	91.24%
16	91.31%	90.97%	91.17%	91.94%	91.09%	91.22%
Best	91.52%	91.14%	91.27%	**92.13%**	91.56%	91.37%
Baseline	88.46%					

TABLE V
RESULTS OF ALL THE METRIC LEARNING ALGORITHMS

All the methods managed to outperformed the simple distance metric, revealing the effectiveness of task-dependent Mahalanobis distance. And the whole procedure of all the methods were finished within an hour, showing its efficiency.

Furthermore, semi-hard mining obviously outperformed all the other methods. We figure that it is because for one illegal pair, semi-hard mining punishes it multiple times, while other methods only punish it one time. In other words, it weights the loss of the illegal pairs that are harder to rectify, while other methods treat all the illegal pairs equally. Therefore, semi-hard mining tends to present a better performance.

The methods involved in our project can be divided into three different categories. LMNN, Contrastive loss and Binomial deviance directly operate the distance; NCA and MCML utilize the tools provided by information theory; Semi-hard mining strategy actually is not a specified algorithm but a margin choosing strategy that can be used by other algorithms. However, in fact, these methods share the same ultimate goal: minimizing the distances between similar samples and maximizing the distances between dissimilar samples. And these algorithms therefore might differ little for this task, except for semi-hard mining, which introduces inherit weights that can bring performance gain.

In addition, just as the dimensionality reduction methods could be used for metric learning, all these methods could also be applied to dimensionality reduction, which could be an interesting field to research on.

F. Comprehensive Comparison

We plot all the methods we performed except Bregman Divergence in Figure 4.

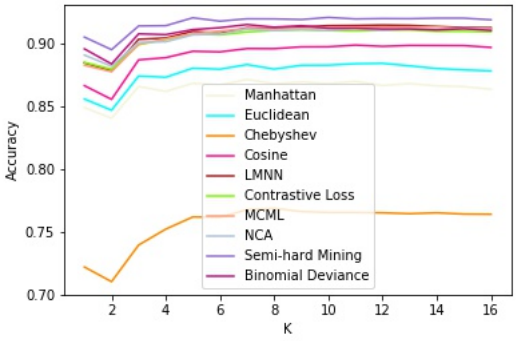


Fig. 4. Accuracy curve of all

We can see that for this task, cosine similarity performed best among all the simple distance metrics. And metric learning methods were all proved to be effective and efficient in accuracy improvement, especially semi-hard mining strategy. Unfortunately, since the data don't possess any probabilistic semantics, the attempt of Bregman Divergence failed.

IV. Further Study

A. Cosine Distance and L2 Normalization

In experiments, we found that cosine distance metric could significantly enhance the KNN performance. Thus in this subsection, we will thoroughly study the positive effect of cosine distance, L2 normalization and its reasons. We'll talk about this topic step by step.

1) Equivalence of Cosine Distance and L2 Normalized Euclidean Distance: Firstly, it can be proved that the cosine distance and Euclidean distance with L2 normalization is equivalent in distance comparison. Or formally, given three samples $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$:

$$\begin{aligned} & dist_{cos}\left(\boldsymbol{x}, \boldsymbol{y}\right) < dist_{cos}\left(\boldsymbol{x}, \boldsymbol{z}\right) \\ \Longleftrightarrow\ & dist_{eucl}\left(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}\right) < dist_{eucl}\left(\hat{\boldsymbol{x}}, \hat{\boldsymbol{z}}\right), \end{aligned} \tag{21}$$

where $\hat{\boldsymbol{x}}$ is the L2 normalized $\boldsymbol{x}$:

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}.$$

The proof is evident and also intuitive. The cosine distance can be represented as:

$$\begin{aligned} dist_{cos}(\boldsymbol{x}, \boldsymbol{y}) &= \frac{1}{2}(1 - Sim_{cos}) \\ &= \frac{1}{2}\left(1 - \frac{\boldsymbol{x}^T \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}\right), \\ &= \frac{1}{2}\left(1 - \hat{\boldsymbol{x}}^T \hat{\boldsymbol{y}}\right) \end{aligned}$$

while L2 normalized Euclidean distance:

$$\begin{aligned} dist_{eucl}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) &= \sqrt{(\hat{\boldsymbol{x}} - \hat{\boldsymbol{y}})^T (\hat{\boldsymbol{x}} - \hat{\boldsymbol{y}})} \\ &= \sqrt{\hat{\boldsymbol{x}}^T \hat{\boldsymbol{x}} + \hat{\boldsymbol{y}}^T \hat{\boldsymbol{y}} - 2\hat{\boldsymbol{x}}^T \hat{\boldsymbol{y}}}. \\ &= \sqrt{2 - 2\hat{\boldsymbol{x}}^T \hat{\boldsymbol{y}}} \end{aligned}$$

Therefore, we get:

$$dist_{eucl}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = 2\sqrt{dist_{cos}(\boldsymbol{x}, \boldsymbol{y})}.$$

$g(t) = 2\sqrt{t}$ is a monotonically increasing function defined on $[0, +\infty)$, i.e. :

$$\forall t_1, t_2 \in [0, +\infty), \quad t_1 < t_2 \iff g(t_1) < g(t_2).$$

Therefore, based on this basic property of monotonic functions, Equation 21 is proved.

2) Experiments of L2 Normalization: Now that we have proved the equivalence of cosine distance and L2 normalization, we conducted experiments on L2 normalization to further prove our conclusion, as well as to ensure its positive effect. We ran our KNN implementation on L2 normalized AwA2 dataset and the best results among different K's on each metric are shown in Table VI.

K \ Metric	L1	L2	L_∞	cos
1	86.56%	86.69%	73.35%	86.68%
2	85.59%	85.60%	72.20%	85.60%
3	88.24%	88.75%	75.62%	88.75%
4	88.02%	88.92%	76.52%	88.92%
5	88.81%	89.43%	77.43%	89.43%
6	88.63%	89.38%	77.78%	89.38%
7	88.95%	89.64%	78.03%	89.64%
8	88.87%	89.64%	77.99%	89.64%
9	**89.13%**	89.78%	**78.19%**	89.78%
10	88.93%	89.79%	78.19%	89.79%
11	89.00%	**89.93%**	78.16%	**89.92%**
12	88.93%	89.83%	78.02%	89.83%
50	87.20%	88.51%	76.35%	88.51%
100	85.23%	87.02%	74.15%	87.02%
200	82.77%	85.22%	71.08%	85.22%
500	78.32%	81.84%	66.48%	81.84%
1000	74.10%	78.59%	62.25%	78.59%
Best	**89.13%**	**89.93%**	**78.19%**	**89.93%**
Baseline	87.18%	88.46%	76.92%	**89.93%**

TABLE VI

RESULT OF SIMPLE DISTANCE METRICS WITH L2 NORMALIZATION

As expected, the performance of cosine distance and L2 normalized Euclidean distance are identical, which has been theoretically supported in subsubsection IV-A.1. And not surprisingly, the performance of all distance metrics (except cosine, because cosine distance is insensitive to normalization) is enhanced by nearly 2%. Therefore, **the positive effect of cosine distance is actually the effect of L2 normalization**. We'll discuss about L2 normalization in the next part.

Since we observed the good performance of Cosine Distance and the equivalence of Cosine Distance and L2 Normalized Euclidean Distance, we implemented a further experiment on Semi-Hard Mining with data after L2 normalization. As was expected, the performance is much better than the experiment using original data. The result is shown in Figure 5. As we can see from the result, L2 normalization can not only enhance simple distance metric but also metric learning. In other words, the enhancement of L2 normalization has little overlap with that of Mahalanobis distance.

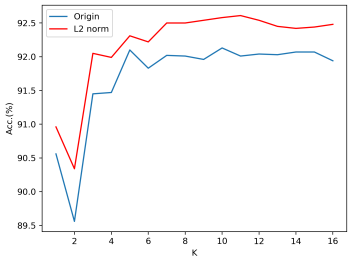


Fig. 5. Result of Semi-Hard Mining using L2 norm

3) L2 Normalization: L2 normalization projects the original samples to the unit hyper-sphere. Obviously, L2 normalization is not a universally good pre-process of dataset. Take Figure 6 as a simple example. After L2 normalization, the two clusters in the example will overlap and be indivisible.

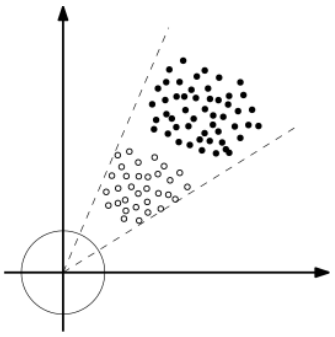
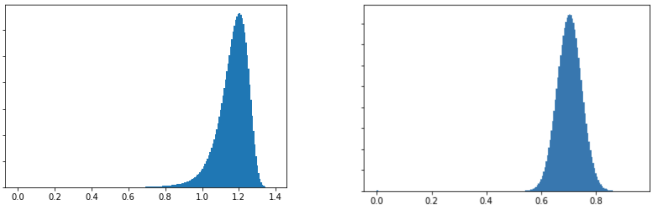


Fig. 6. Unsuitabilility of L2 Normalization

Besides, we also found that our distance distribution is different from uniformly random samples. The skewness of L2 normalized distance distribution is supposed to be 0 but distance of our data set is negative skew. The comparison is shown in Figure 7.



(a) AwA2 data set (b) Uniformly random data

Fig. 7. Distribution of L2 normalized distance of AwA2 and random data

Multiple clues have indicated that the effect of L2 normalization depends on the distribution of data set. So we did further experiments on randomly generated data with Euclidean distance. We tested on the data set with different dimensionality D and generated 10000 samples with 4 clusters for each D. 70% of data set are used as training set and the rest is test set. The hyper-parameter K of KNN is fixed at 5. The results are shown in Table VII.

D	Euclid	L2 normalized Euclid	enhancement
2	93.55%	88.20%	-5.35%
4	84.45%	83.30%	-1.15%
8	83.60%	83.85%	+0.25%
16	83.30%	80.95%	-2.35%
32	68.65%	68.50%	-0.15%
64	56.85%	58.20%	+1.35%
128	48.35%	51.70%	+3.35%
256	41.55%	42.85%	+1.30%
512	35.55%	38.90%	+3.35%
1024	33.35%	35.55%	+2.20%

TABLE VII

EFFECT OF L2 NORMALIZATION ON RANDOM DATA

It's interesting that, generally, **L2 normalization have positive effect on high dimensional data but negative on low dimensional data**, which corresponds with the experiments results before because AwA2 data set is high dimensional (2048). The reason is not clear yet. A reasonable explanation is that high dimensional data tend to distribute on a hyper-surface, hence L2 normalization are less likely to damage the distribution and more possible to smooth the distance distribution.

4) L1 normalization: After analyzing L2 normalization, we are also curious about the effect of L1 normalization. L1 normalization of sample $\boldsymbol{x}$ is defined as:

$$\tilde{\boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_1}.$$

The best performance after L1 normalization is shown in Table VIII. It can be observed that among all these simple distance metrics, L1 normalization enhances L1 distance (Manhattan distance) most significantly, by 1.7% compared to L2 normalization. Naturally, we hypothesized that L_p **normalization can best enhance the performance of** L_p **distance metric**. To attain more clues of this hypothesis, we did experiments on L_3 distance and normalization. The results are appended to Table VIII.

Although L_3 normalization didn't achieve the best accuracy on L_3 distance (perhaps because Euclidean distance greatly outperform other metrics), it achieved maximum gain on L_3 distance compared to baseline while it decreased the accuracy of L_1 and L_2 distance metrics, which partly corresponds to the hypothesis. The research on this phenomenon could also be interesting.

Metric / Norm.	L1	L2	L3
Baseline	87.18%	88.46%	87.12%
L1 Norm	**90.88%**(+3.70%)	90.01%(+1.55%)	88.22%(+1.10%)
L2 Norm	89.13%(+1.95%)	**89.93%**(+1.47%)	88.96%(+1.84%)
L3 Norm	87.02%(-0.16%)	**88.48%**(+0.02%)	87.98%(+0.86%)

TABLE VIII

RESULT OF SIMPLE DISTANCE METRICS WITH L1 NORMALIZATION

B. LMNN and k

Among all the algorithms we discussed, LMNN is the only one involving k as a parameter. We've seen that when $k = 4$, the best performance was not achieved by $K = 4$, and simply discussed the relationship between k and K. We want to explore on how k influence the performance of LMNN. To illustrate this, we trained and evaluated LMNN with different Ks. The results are shown in Table IX. To be more ocular, we also plot the accuracy of different LMNN models in Figure 8.

As shown, k had a strong relation to the classification accuracy. Here we try to explain how the connection worked.

K \ k	1	2	4	8	16
1	88.20%	88.29%	88.51%	88.18%	87.65%
2	87.51%	87.76%	87.86%	87.86%	86.91%
3	90.03%	90.30%	90.39%	89.95%	89.16%
4	89.95%	90.31%	90.49%	90.09%	89.22%
5	90.70%	90.92%	91.02%	90.62%	89.72%
6	90.50%	90.97%	90.86%	90.60%	89.64%
7	90.93%	91.27%	91.30%	91.12%	90.05%
8	90.82%	91.14%	91.24%	90.92%	89.96%
9	91.10%	91.27%	91.37%	91.08%	90.15%
10	91.08%	91.21%	91.47%	90.96%	90.19%
11	91.20%	91.32%	91.49%	91.16%	**90.29%**
12	91.10%	91.30%	**91.52%**	91.07%	90.05%
13	91.23%	91.35%	91.49%	91.12%	90.14%
14	**91.27%**	91.37%	91.41%	91.09%	90.11%
15	91.17%	**91.37%**	91.32%	**91.21%**	90.11%
16	91.20%	91.27%	91.31%	91.08%	90.01%
Best	**91.27%**	**91.37%**	**91.52%**	**91.21%**	**90.29%**

TABLE IX

RESULT OF LMNN WITH DIFFERENT k

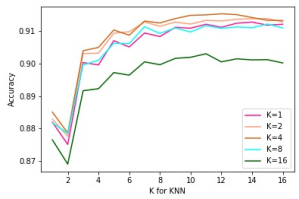


Fig. 8. LMNN accuracy curve

The goal of LMNN is to optimize the leave-one-out cross-validation accuracy when $k = K$, but we can see the best performance was never achieved when $k = K$. It might tell us the choice of k doesn't have much connection with the testing phase. Then what role does k play? We can see that the best performance was achieved when $k = 4$, and there was a huge performance gap between $k = 16$ and others. We think it was related to our setting of batch size. In our data set, there are 50 different labels. With a batch size of 500, every sample has have about 9 target neighbors in a batch. If k was close to 9 or bigger than 9, the supervision we imposed on different classes could be imbalanced since some classes might not have enough target neighbors to supervise with, which finally resulted in relatively poor performance. Therefore, k somewhat implicates the intensity of the supervision we impose on our model. And its choice should be considered with attention to the size of data set (or the mini-batch).

C. Experiments of Semi-Hard Mining Lite

The original paper for Semi-Hard Mining Strategy[4] also used a simplified model, in which only the maximum of positive pairs and the minimum of negative pairs would be taken into account for optimization. We implemented an experiment to explore this model further and it was observed that the lite version of Semi-Hard Mining didn't perform as well as the full version (comparing every two of positive and negative pairs). The result is shown in Figure 9.

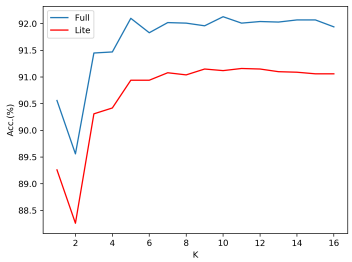


Fig. 9. Result of Semi-Hard Mining Lite

We attributed the different performance between the full version of Semi-Hard Mining and the lite version to the number of pairs that would be optimized in the model, for the lite version only optimized the maximum of positive pairs and the minimum of negative pairs. Thus the advantage of Semi-Hard over other metric learning method is the large number of pairs of distance that is optimized.

V. Conclusion

In this project, we evaluated different distance metrics on the given data and analyzed their performance, then we used several metric learning algorithms to improve the performance, and went through some algorithms' details. Furthermore, considering the close connection between normalization and distance, we performed some exploratory experiments and gave our analysis. Finally, we try to give some general advice. First, simple distance metrics are good choices, providing feasible performance and great generality. Second, L_p normalization might enhance the representational ability of the distance under some certain circumstances. Third, metric learning is great if a task-dependent metric is needed. Fourth, divergence shouldn't be used for non-probabilistic situation.

References

[1] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009.

[2] Raia Hadsell, Sumit Chopra, and Yann Lecun. Dimensionality reduction by learning an invariant mapping. pages 1735 – 1742, 02 2006.

[3] Jacob Goldberger, Geoffrey E Hinton, Sam T. Roweis, and Ruslan R Salakhutdinov. Neighbourhood components analysis. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 513–520. MIT Press, 2005.

[4] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[5] Yongqin Xian, Christoph H. Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning - A comprehensive evaluation of the good, the bad and the ugly. *CoRR*, abs/1707.00600, 2017.

[6] L.M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200 – 217, 1967.

[7] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[8] Amir Globerson and Sam T. Roweis. Metric learning by collapsing classes. In *NIPS*, 2005.

[9] Andreas Klöckner, Nicolas Pinto, Yunsup Lee, B. Catanzaro, Paul Ivanov, and Ahmed Fasih. PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Computing*, 38(3):157–174, 2012.