Project 2: KNN with Different Distance Metrics

Mingquan Feng
517030910373

Mingjie Li
517030910344

Shangning Xu
517030910384

Abstract

Many machine learning algorithms, such as K-Nearest-Neighbor (KNN), heavily rely on the distance metric for the input data patterns.[1] In this project, we conducted our experiment using deep learning features of the AwA2 dataset. We tried common distance metrics (including Chebyshev distance, Euclidean distance, Manhattan distance and cosine distance) for KNN classfication. We also explored the effect of preprocessing. Furthermore, we tried 9 distance learning metrics (including LMNN[2], SDML[3], ITML[4], NCA[5], LFDA[6], LSML[7], MMC[8], RCA[9], and MKLR[10]). All learned metrics outperformed KNN with Eculidean distance, except for MMC. We also analyzed the behaviors and properties of different distance learning algorithms in this report.

1 Introduction

Many machine learning algorithms, such as K-Nearest-Neighbor (KNN), heavily rely on the distance metric for the input data patterns. K-Nearest-Neighbor (KNN) for classification is a type of instance-based learning, or lazy learning algorithm via a voting process. The detailed description of KNN is omitted here. This report mainly focuses on comparing the performance and behaviors of different distance metrics and distance metric learning algorithms.

In this report, we evaluated simple distance metrics (including Chebyshev distance, Euclidean distance, Manhattan distance and cosine distance) and 9 distance metric learning metrics (including LMNN[2], SDML[3], ITML[4], NCA[5], LFDA[6], LSML[7], MMC[8], RCA[9], and MKLR[10]) on pre-extracted deep learning features from the AWA2 dataset. This report is organized as follows: Section 2 includes a comprehensive description of distance metrics and distance metric learning algorithms we explored. In Section 3, we presented experimental results and our analysis of different methods. Section 4 is a summary of this project.

2 Distance Metrics

2.1 Simple Distance and Mahalanobis Distance

Simple Distance denotes data-independent distance functions, e.g. Minkowski Distance and Cosine Distance:

$$\begin{aligned} d_{Minkowski}(\mathbf{x}, \mathbf{y}) &= \left(\sum_i |x_i - y_i|^p \right)^{\frac{1}{p}} \\ d_{cos}(\mathbf{x}, \mathbf{y}) &= 1 - \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \end{aligned}$$

these functions only involves several hyper-parameters, therefore they do not require fitting training data. On the other hand, Mahalanobis Distance is learning a linear embedding $\mathbf{L}$, then it measures L_2

distance between embedded data:

$$\begin{aligned} D(\mathbf{x}, \mathbf{y}) &= \sqrt{(\mathbf{L}\mathbf{x} - \mathbf{L}\mathbf{y})^\top (\mathbf{L}\mathbf{x} - \mathbf{L}\mathbf{y})} \\ &= \sqrt{(\mathbf{x} - \mathbf{y})^\top \mathbf{L}^\top \mathbf{L} (\mathbf{x} - \mathbf{y})} \\ &= \sqrt{(\mathbf{x} - \mathbf{y})^\top \mathbf{M} (\mathbf{x} - \mathbf{y})} \end{aligned}$$

where $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$ is a positive semi-definite (PSD) matrix, called Mahalanobis matrix. The goal of metric learning is to learn a task-specific $\mathbf{M}$. Specifically, there are 2 problem settings:

1. Supervised Learning: The algorithm has access to labelled data. The goal is to put points within same class close together while pushing away points in different classes.
2. Weakly-supervised Learning: The algorithm only has access to tuples of data, without directly knowing labels. For example, given tuples of positive (similar) and negative (dissimilar) pairs, the goal is to put positive pairs close together and negative pairs far away.

Notice that metric learning relates to many applications:

1. K-Nearest Neighbors (KNN): Since Neighbors is defined by distance, accuracy of KNN depends on distance measurement. Therefore metric learning may improve accuracy of KNN.
2. Clustering: metric learning provides a way to bias the clusters towards the intended semantics.
3. Dimensionality Reduction: metric learning may be seen as a way to reduce the data dimension in a (weakly) supervised setting. Notice that the goal of Supervised metric learning is similar with goal of LDA, which is a classic Dimensionality Reduction algorithm.

2.2 Large Margin Nearest Neighbor Metric Learning (LMNN)

LMNN [2] is a supervised model, which is based on two simple intuitions (and idealizations) for robust kNN classification: first, that each training input $x_i$should share the same label y_i as its k nearest neighbors; second, that training inputs with different labels should be widely separated. Therefore in loss function of LMNN, one term penalizes large distances between nearby inputs with the same label, while the other term penalizes small distances between inputs with different labels. Notice that to define notions of "nearby", this framework requires auxiliary information beyond the label y, i.e. *target neighbors*, a priori of nearest neighbors.

Based on intuition and terminology above, we can define loss functions of LMNN:

$$\begin{aligned} \varepsilon_{\text{pull}}(\mathbf{L}) &= \sum_{i \sim j} \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2 \\ \varepsilon_{\text{push}}(\mathbf{L}) &= \sum_{i,j \sim i} \sum_{l} (1 - y_{il}) \left[1 + \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2 - \|\mathbf{L}(\vec{x}_i - \vec{x}_l)\|^2\right]_+ \\ \varepsilon(\mathbf{L}) &= (1 - \mu)\varepsilon_{pull}(\mathbf{L}) + \mu\varepsilon_{push}(\mathbf{L}) \end{aligned}$$

where $j \sim i$ denotes j is target neighbors of i, y_{il} is indicator denoting whether x_i and x_l in same class, $[z]_+ = \max(z, 0)$ is hinge loss.

2.3 Sparse High-Dimensional Metric Learning (SDML)

SDML [3] is a weekly-supervised model, which learns sparse metric via an l_1-penalized log-determinant regularization. The sparsity prior of learning distance metric can be justified from three aspects:

1. in high dimensional input spaces, the off-diagonal elements of concentration matrix (i.e. the inverse of the covariance matrix)are often remarkably small. This observation reflects the sparse correlations be- tween different dimensions and supports a sparse Mahalanobis metric.

2. Compact model is preferred in distance metric learning, since it can help to prevent overfitting.
3. a sparse Mahalanobis distance can be computed very efficiently which is of significant importance to many realistic applications

Therefor, they propose 2 types of constraints, the first type is off-diagonal l_1 norm:

$$\|M\|_{1,\text{off}} = \sum_{i \neq j} |M_{ij}|$$

and the second type of constraint is log-determinant divergence:

$$\begin{aligned} g(M) &= -\log\det(M) \\ D_g\left(M \| M_0\right) &= g(M) - g\left(M_0\right) - \left\langle \nabla_g\left(M_0\right), M - M_0 \right\rangle \\ &= \operatorname{tr}\left(M_0^{-1} M\right) - \log\det M \end{aligned}$$

where M_0 is given prior, and constant term regarding M_0 is ignored in last expression. Using $\mathcal{S}$ to denote similar samples, and $\mathcal{D}$ to denote dissimilar samples, the loss function is defined as following:

$$\begin{aligned} K_{ij} &= \begin{cases} 1, \text{ if } (x_i, x_j) \in \mathcal{S} \\ -1, \text{ if } (x_i, x_j) \in \mathcal{D} \end{cases} \\ \mathcal{L}(\mathcal{S}, \mathcal{D}) &= \frac{1}{2} \sum_{i,j=1}^{n} \left\| A^T x_i - A^T x_j \right\|_2^2 K_{ij} \\ &= \sum_{i,j=1}^{n} \left(x_i^T M x_i - x_i^T M x_j \right) K_{ij} \\ &= \operatorname{tr}\left(X^T M X D\right) - \operatorname{tr}\left(X^T M X K\right) \\ &= \operatorname{tr}\left(X(D-K)X^T M\right) \\ &= \operatorname{tr}\left(X L X^T M\right) \end{aligned}$$

which gives optimization objective of SDML:

$$\begin{aligned} &\min_M \operatorname{tr}\left(M_0^{-1} M\right) - \log\det M + \lambda \|M\|_{1,\text{off}} + \eta \mathcal{L}(S, D) \\ &= \operatorname{tr}\left(\left(M_0^{-1} + \eta X L X^T\right) \cdot M\right) - \log\det M + \lambda \|M\|_{1,\text{off}} \\ &\text{s.t. } M \succeq 0 \end{aligned}$$

2.4 Information Theoretic Metric Learning (ITML)

ITML [4] formulates metric learning as minimizing the differential relative entropy between two multivariate Gaussians under constraints on the distance function, or equivalently minimizing the LogDet divergence subject to linear constraints.

Specifically, they assume data distribution is multivariate Gaussians, and their objective is to minimize KL divergence between learned matrix and prior matrix:

$$\begin{aligned} \min_A \quad & \operatorname{KL}\left(p\left(\boldsymbol{x}; A_0\right) \| p(\boldsymbol{x}; A)\right) \\ \text{subject to} \quad & d_A\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \leq u \quad (i,j) \in S \\ & d_A\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \geq \ell \quad (i,j) \in D \end{aligned}$$

They observe that the equation above be expressed as LogDet Bregman divergence, which generated by convex function $\phi(X) = -\log\det X$ as below:

$$\begin{aligned} D_{\ell\text{d}}\left(A, A_0\right) &= \operatorname{tr}\left(A A_0^{-1}\right) - \log\det\left(A A_0^{-1}\right) - n \\ \operatorname{KL}\left(p\left(\boldsymbol{x}; A_0\right) \| p(\boldsymbol{x}; A)\right) &= \frac{1}{2} D_{\ell\text{d}}\left(A_0^{-1}, A^{-1}\right) \\ &= \frac{1}{2} D_{\ell\text{d}}\left(A, A_0\right) \end{aligned}$$

Therefore, the objective function is:

$$\begin{aligned} \min_{A \succeq 0} \quad & D_{\ell \mathrm{d}}\left(A, A_0\right)=\operatorname{tr}\left(A A_0^{-1}\right)-\log \det \left(A A_0^{-1}\right)-n \\ \text{s.t.} \quad & \operatorname{tr}\left(A\left(\boldsymbol{x}_i-\boldsymbol{x}_j\right)\left(\boldsymbol{x}_i-\boldsymbol{x}_j\right)^T\right) \leq u & (i, j) \in S \\ & \operatorname{tr}\left(A\left(\boldsymbol{x}_i-\boldsymbol{x}_j\right)\left(\boldsymbol{x}_i-\boldsymbol{x}_j\right)^T\right) \geq \ell & (i, j) \in D \end{aligned}$$

It can also be extended by adding slack constraints and kernel function. Notice that ITML is very similar with SDML, with only difference in regularization terms.

2.5 Neighborhood Component Analysis (NCA)

Neighbourhood components analysis (NCA)[5] is a **supervised** learning method for classifying multivariate data into distinct classes according to a given distance metric over the data. NCA aims to learn a distance metric by finding a **linear** transformation of input data such that the average leave-one-out (LOO) classification performance is maximized in the transformed space.

To formally address the problem, let the matrix to learn be A so that the transformed form of $\mathbf{x}$ will be $A\mathbf{x}$. We need to define a differentiable function $f(A)$ indicating the classification performance, and optimize on this function. *I.e.*

$$A^{\star}=\arg \max_{A} f(A) \tag{1}$$

However, the problem of LOO classification accuracy is that the function is not differentiable. In this setting, when elements in A changes smoothly, the set of neighbors for a point i will undergo discrete changes. After all, this is a discrete variable.

The solution is also easy, we can see the neighbors in a "differentiable" way. Rather than considering the k-nearest neighbors at each transformed point in LOO-classification, we'll consider the entire transformed data set as stochastic nearest neighbors. Define these using a softmax function of the squared Euclidean distance between a given LOO-classification point and each other point in the transformed space.

$$p_{ij}=\begin{cases} \frac{\exp(-\|A\mathbf{x}_i-A\mathbf{x}_j\|^2)}{\sum_{k\neq i}\exp(-\|A\mathbf{x}_i-A\mathbf{x}_k\|^2)} & \text{if } j\neq i \\ 0 & \text{if } j=i \end{cases} \tag{2}$$

We do not consider two same points. The meaning of p_{ij} is the probability of $j \in C_i$. Then the probability of correctly classifying data point i is

$$p_i=\sum_{j\in C_i} p_{ij} \tag{3}$$

Define $\mathbf{x}_{ij}=\mathbf{x}_i-\mathbf{x}_j$. Then the objective function and its gradient will be

$$\begin{aligned} f(A) &= \sum_i p_i = \sum_i \sum_{j\in C_i} p_{ij} \\ \frac{\partial f(A)}{\partial A} &= -2A\sum_i \sum_{j\in C_i} p_{ij}\left(\mathbf{x}_{ij}\mathbf{x}_{ij}^{\top}-\sum_k p_{ik}\mathbf{x}_{ik}\mathbf{x}_{ik}^{\top}\right) \\ &= 2A\sum_i\left(p_i\sum_k p_{ik}\mathbf{x}_{ik}\mathbf{x}_{ik}^{\top}-\sum_{j\in C_i} p_{ij}\mathbf{x}_{ij}\mathbf{x}_{ij}^{\top}\right) \end{aligned} \tag{4}$$

2.6 Local Fisher Discriminant Analysis (LFDA)

Local Fisher Discriminant Analysis (LFDA)[6] is one of the best-performing supervised dimensionality reducing metric learning method among the others. It is a **linear supervised** dimensionality

reduction method. It is particularly useful when dealing with multi-modality, where one ore more classes consist of separate clusters in input space.

LFDA is inspired by both Fisher Discriminant Analysis (FDA) and Local Preserving Projection (LPP). We first introduce these two methods and their properties, and then introduce the principles of LFDA. The detailed derivation of FDA and LPP are omitted here. Only basic ideas and formulas are presented.

- **Fisher discriminant analysis (FDA)** is a popular choice to reduce the dimensionality of the original data set. It maximizes between-class scatter and minimizes within-class scatter. But it lacks some considerations for multi-modality. Multi-modality refers to the case where one certain class of data points has two or more clusters.

  Given the dataset $\{\mathbf{x}_i, y_i\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, 2, \ldots, l\}$. Let $S^{(w)}$ and $S^{(b)}$ be the within-class scatter matrix and the between-class scatter matrix defined by the following:

$$S^{(w)} = \sum_{i=1}^{l} \sum_{j:y_j=i} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^\top \tag{5}$$

$$S^{(b)} = \sum_{i=1}^{l} n_i(\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^\top \tag{6}$$

  where $\boldsymbol{\mu}$ is the mean of all data points, $\boldsymbol{\mu}_i$ is the mean of the i^{th} class data points, and n_i is the number of data points belonging to the i^{th} class. FDA aims to find the transformation matrix that satisfies

$$T_{FDA} = \arg\max_{T \in \mathbb{R}^{d \times m}} \; tr\left[(T^\top S^{(w)} T)^{-1}(T^\top S^{(b)} T)\right] \tag{7}$$

- **Local Preserving Projection (LPP)** aims at preserving the local structure of the data. It keeps nearby data pairs in the original data space close in the embedding space so that multimodal data can be embedded and its local structure will not be lost.

  Given the dataset $\{\mathbf{x}_i, y_i\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$. Let $A \in \mathbb{R}^{n \times n}$ be the affinity matrix where its A_{ij} element is the affinity between two points $\mathbf{x}_i$ and $\mathbf{x}_j$. We define A as follows.

$$A_{ij} = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ is a } k\text{-NN of } \mathbf{x}_j \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

  LPP aims to find the transformation matrix that satisfies

$$T_{LPP} = \arg\min_{T \in \mathbb{R}^{d \times m}} \; \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \|T^\top \mathbf{x}_i - T^\top \mathbf{x}_j\|^2 \tag{9}$$

  To avoid the trivial solution $A = O$, T should satisfy $T^\top X D X^\top T = I$ where D is a diagonal matrix that satisfy $D_{ii} = \sum_{j=1}^{n} A_{ij} = k$.

Thus **Local Fisher Discriminant Analysis (LFDA)** is supervised because the "FDA part" is a supervised algorithm. LFDA combines the concepts for both FDA and LPP to define our local Fisher discriminant analysis. The local within- and between-class scatter matrix is defined as

$$S^{(w)} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij}^{(w)} (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^\top \tag{10}$$

$$S^{(b)} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij}^{(b)} (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^\top \tag{11}$$

where $W_{ij}^{(w)} = \begin{cases} \frac{A_{ij}}{n_l}, & y_i = y_j = l \\ 0, & \text{otherwise} \end{cases}$ and $W_{ij}^{(b)} = \begin{cases} A_{ij}\left(\frac{1}{n} - \frac{1}{n_l}\right), & y_i = y_j = l \\ \frac{1}{n}, & \text{otherwise} \end{cases}$. Intuitively, we use the above equations to weight the values for each pair of samples that belong to the same class. Namely, for sample pairs that are far apart from each other in the same class, we give less weight/influence on the two local scatter matrices. The LFDA transformation matrix T is then defined as the following:

$$T_{LFDA} = \arg\max_{T \in \mathbb{R}^{d \times m}} tr\left[(T^\top S^{(w)} T)^{-1}(T^\top S^{(b)} T)\right] \tag{12}$$

There is a demo to show the behavior of FDA, LPP and LFDA in Figure 1. For the simplest data set depicted in (a), both FDA and LPP nicely separate the samples in different classes ('$\circ$' and '$\times$') from each other. For the data set depicted in (b), FDA still works well, but LPP mixes samples in different classes into a single cluster. This is caused by the unsupervised nature of LPP. On the other hand, for the data set depicted in (c), LPP works well but FDA collapses the samples in different classes into a single cluster. The reason for the failure of FDA is that the 'levels' of the between-class scatter and the within-class scatter are not evaluated in an intuitively natural way because of the two separate clusters in '$\circ$'-class. Since LFDA combined the merits of both algorithms, it performs well in all the three cases.

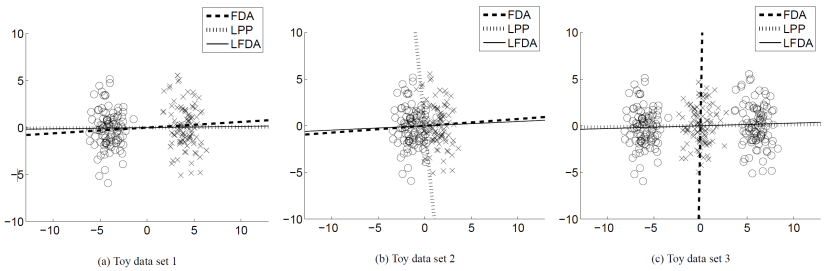


Figure 1: Visualization of the behavior of FDA, LPP and LFDA.

2.7 Least Squares Metric Learning (LSML)

Least Squares Metric Learning (LSML)[7] proposes a simple, yet effective, algorithm that directly learns the Mahalanobis matrix. It is a **weakly-supervised** algorithm that requires information of labels but does not use the full information obtained from labels.

It is an algorithm based on learning on quadruplets. The semantics of a quadruplet $< a, b, c, d >$ is that the first two elements are closer to each other compared with the latter two, *i.e.* $d(a, b) < d(c, d)$. In distance metric learning, we can randomly sample 4 data points from the given data $\{\mathbf{x}_i, y_i\}_{i=1}^n$, two belonging to the same class and the other two not. Formally, we can define the constraint set as follows.

$$\mathcal{C} = \{(\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d) : d(\mathbf{x}_a, \mathbf{x}_b) < d(\mathbf{x}_c, \mathbf{x}_d)\} \tag{13}$$

For each quadruplet, we can define the loss function as $H(d_M(\mathbf{x}_a, \mathbf{x}_b) - d_M(\mathbf{x}_c, \mathbf{x}_d))$ where M is the Mahalanobis matrix to learn. And $H(\cdot)$ is defined as a quadratic hinge loss function: $H(x) = \max(x, 0)^2$. The total loss is defined as

$$\min_M \left[D_{ld}(M, M_0) + \sum_{(\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d) \in \mathcal{C}} H(d_M(\mathbf{x}_a, \mathbf{x}_b) - d_M(\mathbf{x}_c, \mathbf{x}_d)) \right] \tag{14}$$

The first term is the regularization term where M_0 is a prior metric matrix set as identity by default. $D_{ld}(\cdot, \cdot)$ is the Logdet divergence: $D_{ld}(M, M_0) = \text{tr}(MM_0) - \text{logdet}(M)$.

2.8 MMC

MMC, proposed by Xing et al. [8], is one of the earliest attempt to devise a learning algorithm under the scenario of weakly-supervised metric learning and its paper is still widely cited to this day. MMC has a simple mathematical formulation: it minimizes the sum of squared distances between similar points, while enforcing the sum of distances between dissimilar ones to be greater than one. The optimization problem is

$$\begin{aligned} \text{minimize} \quad & \sum_{(x_i,x_j)\in S} ||x_i - x_j||_A^2 \\ \text{subject to} \quad & \sum_{(x_i,x_j)\in D} ||x_i - x_j||_A \geq 1 \\ & A \succeq 0 \end{aligned}$$

This leads to a convex and, thus, local-minima-free optimization problem that can be solved efficiently. However, the algorithm involves the computation of eigenvalues, which is the main speed-bottleneck. Since it has initially been designed for clustering applications, one of the implicit assumptions of MMC is that all classes form a compact set, i.e., follow a unimodal distribution, which restricts the possible use-cases of this method.

2.9 Relevant Component Analysis (RCA)

Shental et al. pioneered *adjustment learning*, a form of weakly-supervised learning, where training data are grouped into "chunklets". Samples within a chunklet belong to the same class. Adjustment learning has its applications in scenarios like retrieval of video clips or tracking an object in surveillance video feed, where consecutive frames are associated with the same object.

Under supervised training, we can find the desired rescaling transformation W that transform data into a whitened space, where nearest neighbor retrieval based on the Euclidean distance is equivalent to maximum likelihood estimation. Let $x \in \Omega$ denote the sampled data, and $|\Omega|$ denote the size of the sample. Similarly, let $|\Omega_m|$ denote the size of the sample from class m where $\bigcup \Omega_m = \Omega$. Let the random variable C_m denote the distribution of the mth class, and M denote the number of classes. For simplicity we also assume that C_m is distributed normally, i.e. $C_m \sim N(\mu_m, \Sigma_m)$, where μ_m denotes the mean of the class and Σ_m denotes its covariance matrix. Let S_W denote the within-class scatter of the data set, defined by

$$S_W = \frac{1}{|\Omega|} \sum_{m=1}^{M} |\Omega_m| \hat{\Sigma}_m$$

where

$$\hat{\Sigma}_m = \frac{1}{|\Omega_m|} \sum_{x \in C_m} (x - \mu_m)(x - \mu_m)^T$$

Assume first that the covariance matrix of all the classes is identical, i.e. $\Sigma_m = \Sigma, \forall m$, and therefore $S_W = \Sigma$ for large enough sample. In this case the optimal rescaling transformation is the whitening transformation W defined as

$$W = V\Lambda^{-\frac{1}{2}}$$

where V is the orthogonal matrix of eigenvectors of S_W, and Λ is its corresponding matrix of singular values. V and Λ may be obtained from the singular value decomposition of $S_W = V\Lambda V^T$.

Environmental conditions are considered in the random noise G, with the following assumptions:

- G is additive with 0 mean and non-isotropic covariance.
- G is independent of C_m, and has an identical effect on all classes.
- The variability in G is relatively large compared to C_m, i.e $|\Sigma_G| > \Sigma_m|$.

More specifically, assume that G is distributed normally as $G \sim N(0, \Sigma_G)$ where $\Sigma_G \neq I$. For class m, the distribution of the observed measurements is:

$$X_{\text{Observable}} = C_m + G$$

The distribution of the mth class is therefore normal, with mean μ_m and covariance $\Sigma_m + \Sigma_G$. Under the assumption that $|\Sigma_G| > \Sigma_m|$, the class distributions are dominated by $|\Sigma_G|$. This effectively brings us back to the previous case where $\Sigma_m \approx \Sigma_G$, and therefore

$$S_W \approx \hat{\Sigma}_G$$

Let H_n denote the sample of the nth chunklet where $H_n = \Omega$. We assume that the number of chunklets N is much larger than the number of classes M. We further assume that $\forall n, H_n \subseteq C_m$ for some unknown label m. Let $\hat{\mu}_n$ denote the mean of chunklet H_n. We define the chunklet scatter matrix S_{ch} as:

$$S_{\text{ch}} = \frac{1}{|\Omega|} \sum_{n=1}^{N} |H_n| \text{Cov}(H_n) = \frac{1}{|\Omega_n|} \sum_{n=1}^{N} \sum_{j=1}^{|H_n|} (x_n^j - \hat{\mu}_n)(x_n^j - \hat{\mu}_n)^T$$

Under the ideal conditions where each chunklet is chosen randomly from the appropriate class and is large enough, it can be shown that $\text{Cov}(H_n)$ approaches $\Sigma_{m_n} + \Sigma_G$, where m_n is the class label of chunklet H_n. In this case $S_{\text{ch}} = S_W$. In reality, however, data points in the same chunklet tend to have stochastic dependency, and therefore cannot be assumed to be independently sampled from the data with distribution identical to the corresponding class distribution.

RCA learns a full rank Mahalanobis distance metric based on a weighted sum of in-chunklets covariance matrices. It applies a global linear transformation to assign large weights to relevant dimensions and low weights to irrelevant dimensions. Those relevant dimensions are estimated using "chunklets", subsets of points that are known to belong to the same class.

For a training set with n training points in k chunklets, the algorithm is efficient since it simply amounts to computing

$$\mathbf{C} = \frac{1}{n} \sum_{j=1}^{k} \sum_{i=1}^{n_j} (\mathbf{x}_{ji} - \hat{\mathbf{m}}_j)(\mathbf{x}_{ji} - \hat{\mathbf{m}}_j)^T$$

where chunklet j consists of $\{\mathbf{x}_{ji}\}_{i=1}^{n_j}$ with a mean $\hat{m}j$. The inverse C^{-1} is used as the Mahalanobis matrix.

2.10 Metric Learning for Kernel Regression (MLKR)

Kilian and Gerald [10] proposes using results of kernel regression as objective of metric learning. As the name implies, kernel regression makes prediction of a training sample based on a weighted average of its surrounding training samples. It is a kernel method because the weights are computed by applying kernel functions to distances. The metric is learned by minimizing the leave-one-out regression error. More precisely, the estimated y_i for the ith training sample is

$$\hat{y}_i = \frac{\sum_{j \neq i} k_{ij} y_j}{\sum_{j \neq i} k_{ij}}$$

where k_{ij} are weights estimated by

$$k_{ij} = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{d(x_i, x_j)}{\sigma^2})$$

$d(x_i, x_j)$ is the Mahalanobis distance to learn. The optimization target is the loss function

$$L = \sum_i (y_i - \hat{y}_i)^2$$

where $y_i - \hat{y}_i$ is the leave-one-out regression error for the ith training sample. The loss then can be optimized using standard techniques like gradient descent.

Figure 2 illustrates kernel regression under varying distance metrics. It can be observed that the learned metric adapts to the direction of the target function.

A naive computation of the gradient requires to compute the outer product between all pair of vectors. This would result in a complexity of order $O(D^2 n^2)$. Kilian and Gerald suggests that the gradient can be approximated by making use of the fast decay of k_{ij}. Only the $c = 1000$ nearest neighbors of each vector x_i are considered. Also, weights of very small values ($k_{ij} < e^{-34}$) are disregarded.

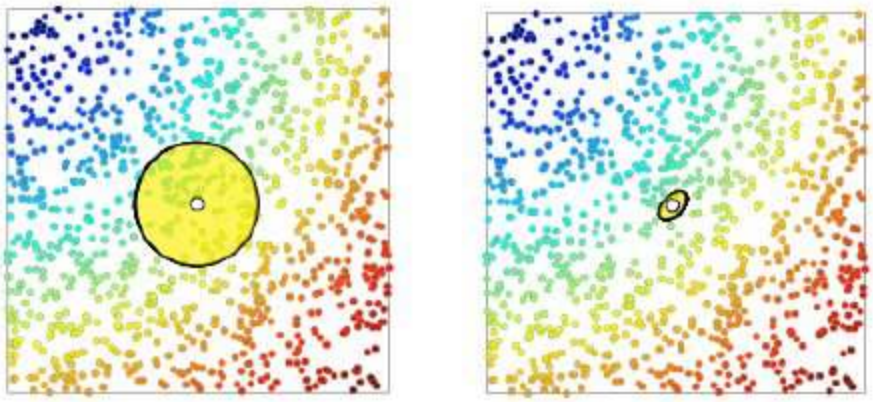


Figure 2: An illustration of kernel regression under varying distance metrics. The color of the points represents the function value. The circle shows the radius that encapsulates 95% of the weights. The function value is estimated at a test point at the center. *Left*: With the use of the Euclidean Distance metric, the kernel is spherical and ignores the present structure that points along the diagonal share similar function values. *Right*: After training, under the Mahalanobis metric, the kernel function follows the direction of the target function. The radius has also shrunk and has therefore adapted to the relatively densely sampled and noiseless training samples.

3 Experiment

3.1 Dataset

Download Animals with Attributes (AwA2) dataset[1]. This dataset consists of 37322 images of 50 animal classes with 2048-dimensional pre-extracted deep learning features for each image. We randomly split the images in each category into 60% for training and 40% for testing, using stratified sampling. We also use 5-fold cross-validation within the training set to select optimal parameters.

3.2 Implementation

We use *metric-learn* [11] package[2] to implements all metric learning algorithms. The basic training framework is *sklearn*.

3.3 Pre-processing

In order to reduce time complexity, we preprocess input data using LDA, and reduce feature dimension to 49. In experiments of simple distance, we find that such pre-process can slightly improve classification performance. Also, the dimension reduction gives 40x running time reduction in our experiments of simple distance KNN.

3.4 Simple Distance

The experiment result is as Table 1 and Figure 3. From the result we find that LDA preprocess can indeed improve performance by approximately 5%, since LDA projects data with same class closer, which is consistent with motivation of KNN. Among 4 simple functions, the Cosine function performs best when n neighbor is small, and L2 function performs best when when n neighbor is large. Such result is similar as in our previous experiment on PCA, where Cosine kernel outperforms all other kernels when n components is small, but Polynomial kernel performs best when n components is large. Such results suggests that our input data may locally distributed in a circular way, where cosine function can best evaluate data relationship.

[1] https://cvml.ist.ac.at/AwA2/

[2] https://github.com/scikit-learn-contrib/metric-learn

Table 1: Accuracy of KNN with different N neighbors and distance functions.

K	Cheb	L1	L2	Cos	L2_NoLDA
5	91.47%	92.13%	92.38%	**92.63%**	87.36%
7	91.77%	92.20%	92.38%	**92.78%**	87.23%
9	91.85%	92.35%	92.58%	**92.71%**	87.13%
11	91.96%	92.35%	92.62%	**92.77%**	87.00%
13	91.99%	92.40%	92.64%	**92.78%**	86.90%
15	92.03%	92.28%	**92.70%**	**92.70%**	86.63%
17	92.03%	92.34%	**92.73%**	92.67%	86.46%
19	92.02%	92.38%	**92.75%**	92.65%	86.26%
21	91.99%	92.33%	**92.75%**	92.64%	85.98%
23	91.99%	92.31%	**92.70%**	92.62%	85.67%

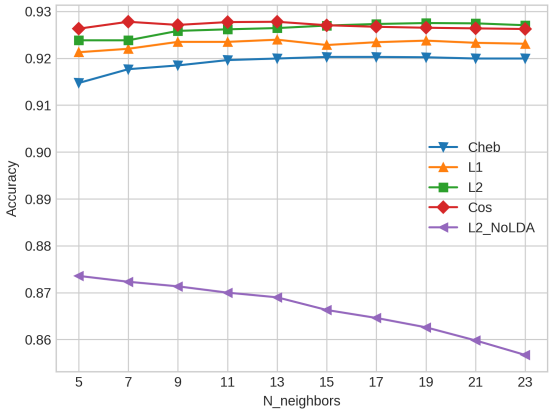


Figure 3: Accuracy of KNN with different N neighbors and distance functions.

3.5 LMNN

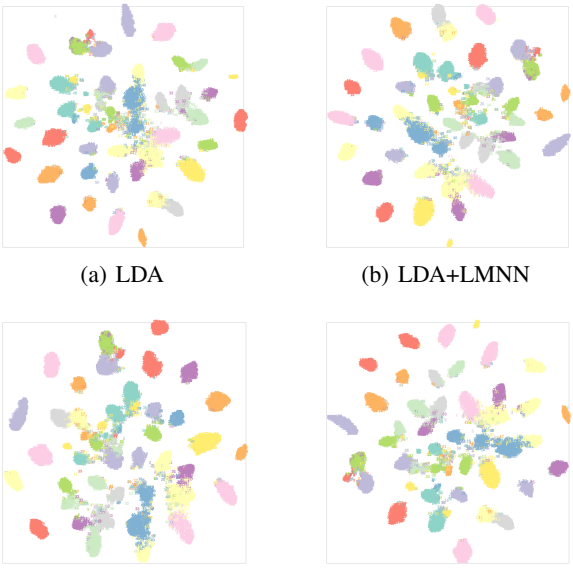
In LMNN, we set initial matrix $init = identity$, $n\_components = 49$ and set other parameters as default. We inspect the accuracy change with regard to k, number of neighbors to consider, and result is as in Table 2 and Figure 5. The table and figure illustrate that LMNN is slightly worse than simple function baseline by 0.1%, we suggest 2 reasons:

1. Effect of initial distance. LMNN needs to first determine target neighbors before learning, and these target neighbors are calculated based on given initial distance. A possible improvements will be using EM algorithm to avoid negative effect of bad initialization.
2. Effect of LDA. Since we pre-process data using LDA, which is also a general type of metric learning. It is possible that LDA itself is enough for metric learning task, and LMNN can not find a better transformation to apply.

To support our second assumption, we use t-SNE to visualize transformed data and original data in Figure 4. We can intuitively judge that after LDA, the data points are separated well enough. At least, LDA pre-process is enough for KNN accuracy.

3.6 ITML and SDML

Both ITML and SDML is weakly-supervised algorithm, therefore we sample $Num\_constraints$ positive pairs and $Num\_constraints$ negative pairs from our supervised dataset. Since sklearn graphical lasso algorithm may not converge in SDML, we set $balance\_param = 1e^{-5}$, $sparsity\_param = 1e^{-5}$ in SDML. All other hyper-parameters are default. The accuracy with regard to $Num\_constraints$ is as in Table 3 and Figure 7. We have following observations:



(a) LDA (b) LDA+LMNN

(c) LDA+ITML (d) LDA+SDML

Figure 4: 2-D visualization of data using t-SNE.

Table 2: Accuracy of LMNN with regard to different K.

K	LMNN
5	**92.28%**
7	92.22%
9	92.12%
11	92.20%
13	92.18%

1. SDML outperforms ITML in all sets of number constraints, which suggests sparse constraints can indeed improve performance.
2. Also, both SDML and ITML reaches peak at $Num\_constraints = 3000$, after which increasing $Num\_constraints$ leads to worse performance. This might be result of conversion from supervised data to weakly supervised data. Such conversion may losing information, or equivalently, adding noise. Therefore increasing $Num\_constraints$ may increase noisy data, and lead to worse performance.
3. The best performance of SDML is almost same with best performance L_2 distance. The reason may be also LDA preprocess, as in Figure 4.
4. In our experiments, the running time of ITML is around 100x of SDML. This result is consistent with theoretical analysis, since sparsity can enhance speed.

To visualize the difference of ITML and SDML, we visualize the learned matrix in Figure 6, which illustrates the sparsity of SDML.

3.7 NCA

In NCA, we set the parameter *n_component* by default. The transformed data point $A\mathbf{x}_i$ is a 2048-dimensional feature vector same as the dimension of $\mathbf{x}_i$. We trained the NCA model on training data and tested the performance on testing data. For fair comparison, we did not do any pre-processing. The baseline is set as the result of direct k-NN without LDA pre-processing. The results are shown in Table 4. As we can see, there is a 3.7% boost with respect to the classification accuracy compared with the baseline. There are two main reasons for this improvement:

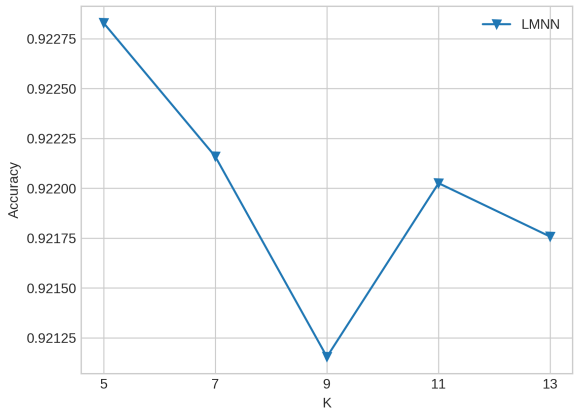
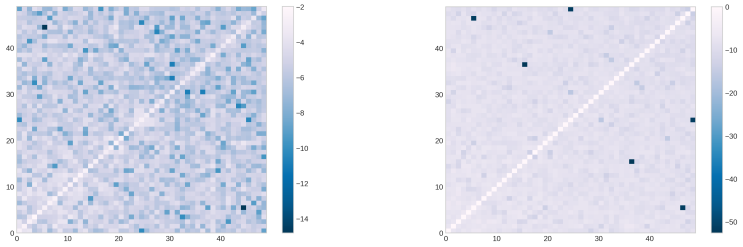


Figure 5: Accuracy of LMNN with regard to different K.



(a) ITML (b) SDML

Figure 6: Visualization of learned matrix from ITML and SDML. The visualized matrix is $log(abs(M))$, where M denotes Mahalanobis matrix. Intuitively SDML is more sparse then ITML, since the off-diagonal value of ITML have more non-zero value.

1. NCA is a supervised algorithm. When training the model, it only force data points in the same class to be closer, instead of preserving local information.
2. The distribution of the training data is similar to that of the testing data. Therefore, the model learned on training data can be generalized to testing data.

From the formulation, we can find NCA has very similar core idea to t-SNE[12]. Actually, both of the two algorithms are proposed by Hinton, in 2005 and 2008 respectively. In NCA and t-SNE, the relationship between data pairs are formulated as probability distribution. They both force data points in the same class to be closer.

Actually, the objective function in Equation 4 is equivalent to minimizing the KL divergence between the real class distribution and the predicted class distribution. To expatiate, the real class distribution $\mathbf{q}_i$ of the data point i is a one hot vector, in which the y_i-th entry of $\mathbf{q}_i$ is 1 and others are 0. The predicted class distribution $\mathbf{p}_i$ of data point i is defined as $(\mathbf{p}_i)_j = \sum_j p_{ij}$, which denotes the probability of i classified as the j-th class. Then $p_i = (\mathbf{p}_i)_{y_i}$ Therefore, we have

$$\max_A \sum_i p_i \Leftrightarrow \max_A \sum_i \log p_i = \sum_i KL(\mathbf{q}_i \| \mathbf{p}_i) \tag{15}$$

By visualizing the transformed data, we can also find the similarity. Figure 8 shows the visualization result. We can find NCA also "cluster"s data points in the same class, though not as good as the performance of t-SNE. From this, we can find the similarity of the two algorithms.

Table 3: Accuracy of ITML and SDML with regard to different $Num\_constraints$.

Num_constraints	ITML	SDML
1000	91.79%	92.34%
3000	**91.99%**	**92.37%**
5000	91.53%	92.31%
7000	91.52%	92.34%
9000	91.64%	92.31%
11000	91.39%	92.35%
13000	91.31%	92.29%
15000	91.37%	92.24%
17000	91.35%	92.22%
19000	91.67%	92.20%

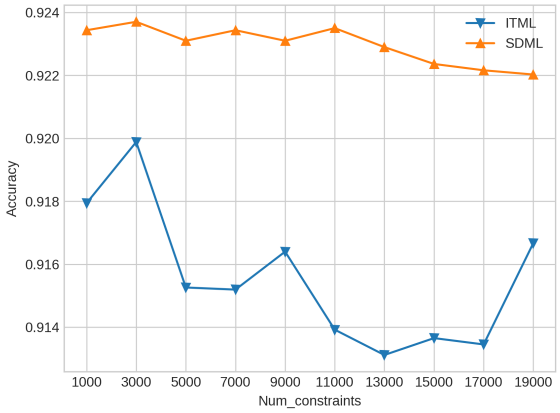


Figure 7: Accuracy of ITML and SDML with regard to different $Num\_constraints$.

3.8 LFDA

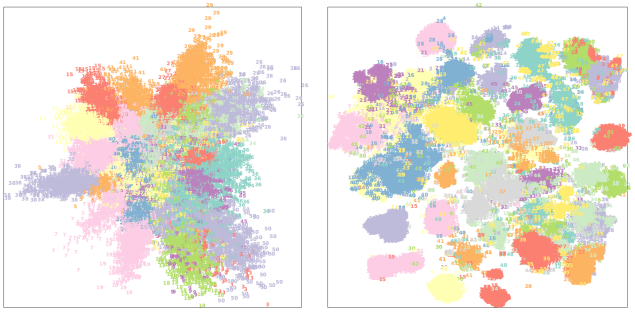
In LFDA, we trained the LFDA model on training data and tested the performance on testing data. For fair comparison, we did not do any pre-processing. The baseline is set as the result of direct k-NN without LDA pre-processing. The results are shown in Table 5. As is suggested in the official document of the *metric_learn* library, LFDA is also a dimensionality reduction method. Therefore, we tried different configurations of the parameter *n_component* from $\{2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$. Also note that the training speed of LFDA is also fast.

The results are shown in Table 5. As we can see, higher dimensionality does not necessarily lead to higher accuracy. This is probably because of the difficulty in training and the over-sized parameter space, leading to a sub-optimal solution. The highest accuracy occurs when the reduced dimension is 64, with a 5% boost compared with the baseline.

We visualized the transformed data using t-SNE in Figure 9. As we can see, the quality of the 64-dim transformed data is higher than that of 2048-dim. Though it seems that the clusters in Figure 9(b) distributes apart, in the center they overlap to each other. This is consistent with the result in Table 5.

Table 4: Accuracy of NCA with respect to different K

Method \ K	5	7	9	11	13	15	17	19	21	23
NCA	90.98%	**90.92%**	90.98%	90.94%	90.85%	90.70%	90.61%	90.59%	90.45%	90.31%
Baseline	87.36%	87.23%	87.13%	87.00%	86.90%	86.63%	86.46%	86.26%	85.98%	85.67%



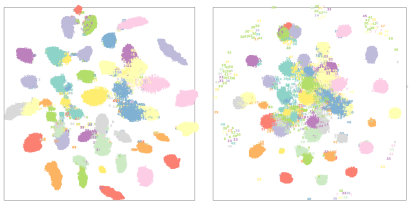
(a) 2-dim NCA (b) 2-dim t-SNE

Figure 8: Comparison of the transformed data of t-SNE and NCA.

Table 5: Accuracy of LFDA with respect to different K and reduced dimension m

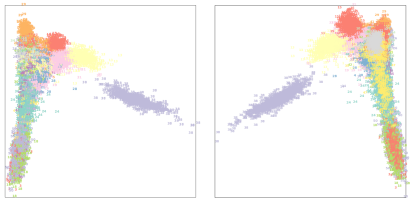
Method \ K	5	7	9	11	13	15	17	19	21	23
LFDA (m=2)	28.22%	28.81%	29.77%	30.46%	30.85%	31.07%	31.42%	31.41%	31.60%	31.83%
LFDA (m=4)	50.11%	51.20%	51.68%	52.13%	52.46%	52.72%	53.04%	53.15%	53.41%	53.25%
LFDA (m=8)	64.08%	64.89%	65.76%	65.61%	66.15%	66.00%	66.23%	66.38%	66.37%	66.41%
LFDA (m=16)	78.93%	79.60%	79.89%	80.00%	79.95%	80.13%	80.27%	80.06%	80.06%	80.13%
LFDA (m=32)	88.43%	88.61%	88.87%	88.97%	88.99%	88.93%	89.10%	89.06%	89.10%	88.93%
LFDA (m=64)	91.82%	**92.12%**	92.08%	92.04%	92.00%	92.14%	92.18%	92.14%	92.14%	92.12%
LFDA (m=128)	91.94%	91.94%	91.96%	92.02%	92.04%	92.02%	91.96%	91.94%	91.92%	91.98%
LFDA (m=256)	91.43%	91.47%	91.58%	91.62%	91.63%	91.50%	91.53%	91.48%	91.48%	91.43%
LFDA (m=512)	90.60%	90.72%	90.76%	90.90%	90.84%	90.75%	90.73%	90.67%	90.58%	90.62%
LFDA (m=1024)	88.71%	88.71%	88.73%	88.60%	88.76%	88.63%	88.53%	88.49%	88.34%	88.15%
LFDA (m=2048)	86.01%	86.25%	86.24%	86.14%	86.02%	85.99%	85.83%	85.75%	85.48%	85.48%
Baseline	87.36%	87.23%	87.13%	87.00%	86.90%	86.63%	86.46%	86.26%	85.98%	85.67%

The behavior of LFDA is similar to that of LDA, as is shown in Figure 10. This indicates a strong relationship between LDA and LFDA.



(a) 64-dim LFDA (b) 2048-dim LFDA

Figure 9: Comparison of the transformed data of different dimensions via t-SNE.



(a) 2-dim LFDA (b) 2-dim LDA

Figure 10: Comparison of the transformed data of LDA and LFDA (w/o t-SNE)

3.9 LSML

In LSML, we set the parameter *n_constraint* by default. We have checked the source code of LSML[3], and found this parameter is set to $20 \times |\mathcal{Y}|^2$. For fair comparison, we did not do any pre-processing. The baseline is set as the result of direct k-NN without LDA pre-processing. The results are shown in Table 4. As we can see, there is a 2.3% boost with respect to the classification accuracy compared with the baseline.

This improvement is not large. We think there are two main reasons:

[3] https://github.com/scikit-learn-contrib/metric-learn/blob/master/metric_learn/lsml.py

Table 6: Accuracy of LSML with respect to different K

Method \ K	5	7	9	11	13	15	17	19	21	23
LSML	89.31%	**89.51%**	89.36%	89.22%	88.95%	88.73%	88.71%	88.54%	88.29%	88.19%
Baseline	87.36%	87.23%	87.13%	87.00%	86.90%	86.63%	86.46%	86.26%	85.98%	85.67%

1. LSML is a weakly supervised algorithm, which does not take full information of the label domain. For example, if the data appears to have a "Swiss roll" distribution, the use of Euclidean distance for relative comparison will be wrong.
2. Maybe the constraint set generated is not big enough. *I.e.* the number of samples in inadequate, leading to a sub-optimal solution.

3.10 MMC

We use the supervised version `MMC_Supervised` from the `metric-learning` project to utilize labels from the AwA2 dataset. As a supervised version of an unsupervised algorithm, its `n_constraint` is also set to $20|\mathcal{Y}|$ as in the case of LSML. The accuracy with respect to different K, the number of nearest neighbors considered in KNN, is shown in Table 7.

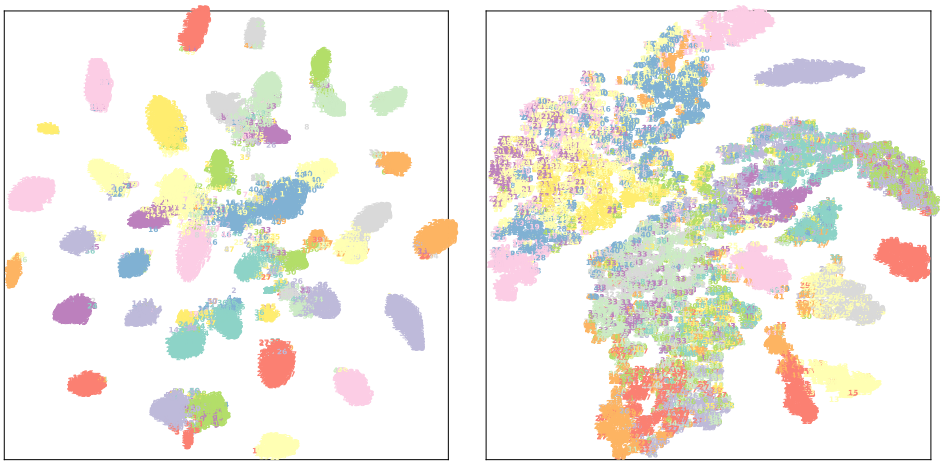


Figure 11: Visualization of all classes with t-SNE. The left side is test data, with their dimensionality reduced to 49 by LDA, and the right side is test data, after dimensionality reduction, projected into the space of MMC's Mahalanobis matrix.

By comparision, MMC have significantly low accuracy on test set compared to all other methods. We hypothesizes that, as mentioned in Section 2.8, the performance of MMC suffers from its assumption of unimodal distribution for all classes. To verify our hypothesis, we extract the Mahalanobis matrix and project test data into this space. Projected test data are visualized with t-SNE. For comparison, unprojected data are also visualized with t-SNE. To illustrate the effects of MMC on each cluster, each cluster is plotted separately on different figures. The plots for clusters with labels from 1 to 4 are shown in Figure 14. It can be observed that, in order to minimize within cluster similarity to adjust for a few outliers, the MMC algorithm makes a trade off between cluster "tightness" and overall closeness. This is detrimental for prediction, as it can be seen from Figure 11, where clusters are much more scattered after MMC, leading to an increase of classification error.

Table 7: Accuracy of RCA, MLKR and MMC with respect to different K

K	1	3	5	7	9	11	13	15	17	19	21
MLKR	89.74%	90.25%	90.56%	90.65%	90.67%	90.69%	90.67%	90.63%	90.6%	90.57%	90.57%
RCA	90.1%	91.3%	91.75%	91.78%	91.79%	91.83%	92.0%	91.94%	91.84%	91.85%	91.83%
MMC	53.85%	56.41%	58.28%	59.45%	59.69%	59.92%	60.1%	60.31%	60.41%	60.31%	60.59%

3.11 RCA and MLKR

AS a weakly-supervised learning algorithm, RCA is converted to a supervised learning algorithm by sampling `num_chunks` chunklets of size `chunk_size`, where training samples within each chunklet belongs to the same unknown class. By comparison, MLKR is a supervised learning algorithm. Due to constraints on computing resources, only 1 0000 samples are used to train MLKR. The accuracy with regard to K is shown in Table 7 and plotted in Figure 12. A setting of 9 for K should be optimal for most algorithms tested, but difference in K only leads to minor performance variations.

Compared with baseline, accuracy of RCA and MLKR is lower, probably due to RCA's inability to fully utilize label information, and constraints on computing resources for MLKR. Their performance is unsuprising, given their similarity in visualization to other methods, as shown in Figure 13.

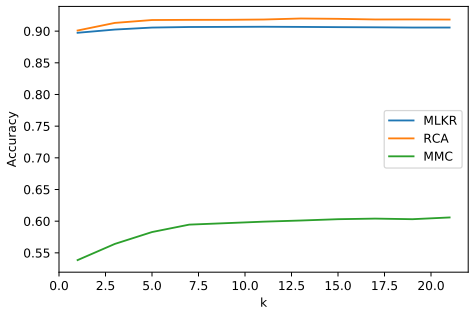


Figure 12: Accuracy of RCA, MLKR and MMC with respect to different K

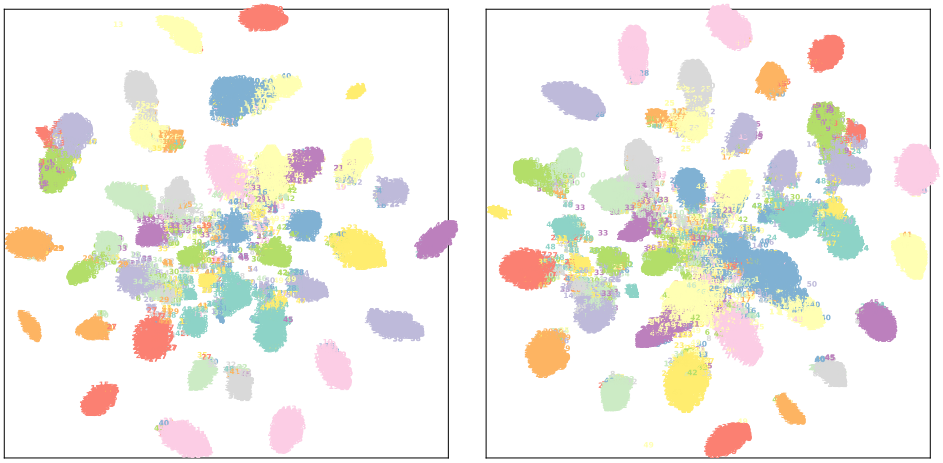


Figure 13: Visualization of RCA and MLKR with t-SNE. The left figure shows RCA.

4 Conclusion

In this project report, we evaluate the performance of KNN classifier with various distance metrics, including simple distance metrics like Euclidean, Chebyshev, L1 distance and cosine distance, and learned metrics through algorithms like LMNN, SDML, ITML, NCA, LFDA, LSML, MMC, RCA and MLKR. Our results show that first, the choice of K, the number of nearest neighbors conisdered in KNN, has a minor effect on performance, and second, LDA boost KNN performance while greatly speed up training and testing, and third, learned metrics give comparable or inferior performance compared with simple metrics. Effects of parameters like n_components, num_constraints on performance are also evaluated and analyzed in the report.

References

[1] Liu Yang and Rong Jin. Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2(2):4, 2006.

[2] 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.

[3] Guo-Jun Qi, Jinhui Tang, Zheng-Jun Zha, Tat-Seng Chua, and Hong-Jiang Zhang. An efficient sparse metric learning in high-dimensional space via l 1-penalized log-determinant regularization. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 841–848, 2009.

[4] Jason V Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216, 2007.

[5] Jacob Goldberger, Geoffrey E Hinton, Sam T Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. In *Advances in neural information processing systems*, pages 513–520, 2005.

[6] Masashi Sugiyama. Dimensionality reduction of multimodal labeled data by local fisher discriminant analysis. *Journal of machine learning research*, 8(May):1027–1061, 2007.

[7] Eric Yi Liu, Zhishan Guo, Xiang Zhang, Vladimir Jojic, and Wei Wang. Metric learning from relative comparisons by minimizing squared residual. In *2012 IEEE 12th International Conference on Data Mining*, pages 978–983. IEEE, 2012.

[8] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems*, pages 521–528, 2003.

[9] Noam Shental, Tomer Hertz, Daphna Weinshall, and Misha Pavel. Adjustment learning and relevant component analysis. In *European conference on computer vision*, pages 776–790. Springer, 2002.

[10] Kilian Q Weinberger and Gerald Tesauro. Metric learning for kernel regression. In *Artificial Intelligence and Statistics*, pages 612–619, 2007.

[11] William de Vazelhes, CJ Carey, Yuan Tang, Nathalie Vauquier, and Aurélien Bellet. metric-learn: Metric Learning Algorithms in Python. Technical report, arXiv:1908.04710, 2019.

[12] Van Der Maaten Laurens and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2605):2579–2605, 2008.

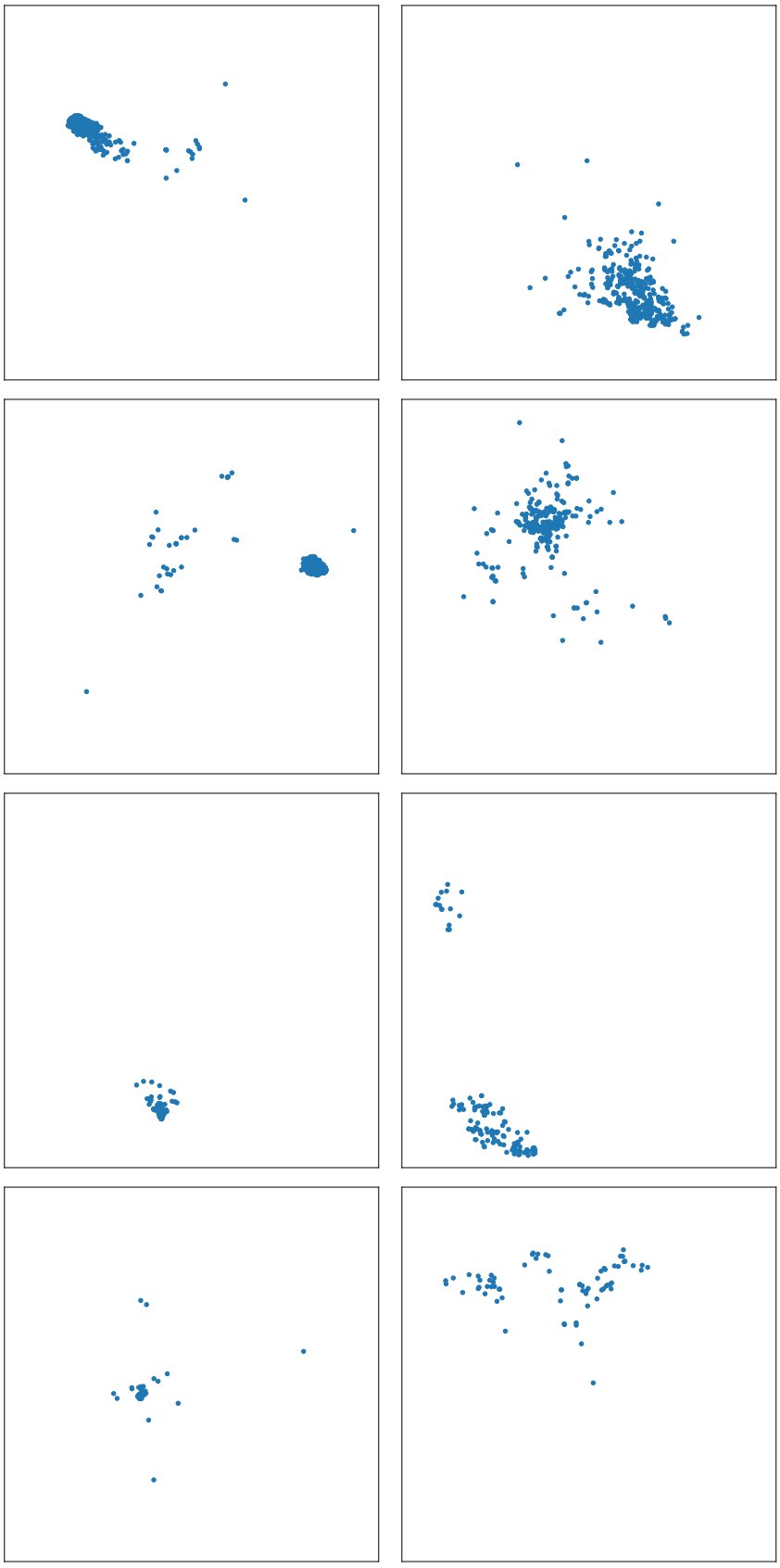


Figure 14: Comparison between test data before and after MMC. The left column shows unprojected test data, and the right column shows projected data. Each row corresponds to class label 1–4, top-down.