Principles of Data Science Project 2 Distance Metrics

Hongzhou Liu
517030910214
deanlhz@sjtu.edu.cn

Xuanrui Hong
517030910227
hongxuanrui.1999@sjtu.edu.cn

Qilin Chen
517030910155
1017856853@sjtu.edu.cn

***Abstract*—The distance metric is an important topic in machine learning. A lot of algorithms rely on a suitable distance metric to perform well, especially k-NN. The choice of k nearest neighbors is closely related to the distance metric. In this project, we tried some classical distance metrics and different metric learning methods to learn some new distance metrics then applied them on k-NN classifier and compared the differences on the classification results.**

***Index Terms*—kNN, Distance Metric, Metric Learning**

I. Introduction

A. k-Nearest Neighbor

K-Nearest Neighbor is a kind of supervised learning method. It can be used for both classification and regression. In the case of classification, the input consists of the k closest training examples in the feature space and the output is a class membership. A certain object is classified by a plurality vote of its neighbors, with it being assigned to the class which is the most common in its k nearsest neighbors. The algorithm is non-parametric [1], which means the model is distribution free or with a specified distribution whose parameters are unspecified. It is also a type of instance-based learning or lazy learning, where the generalization of the training data is delayed until a query is made. In this case, k-NN has no explicit training step and does all computations during testing period. Because we are finding the k nearest neighbors, the distance metric to evaluate "nearest" is significant in k-NN.

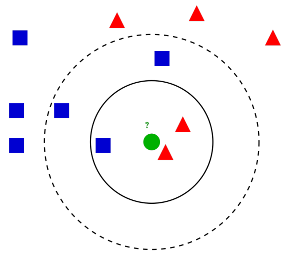


Fig. 1. KNN

B. Distance Metrics

1) Minkowski Distance: Minkowski Distance is a metric in normed vector space. It is a generalization of the well-known Euclidean distance, the Manhattan distance and the Chebyshev distance. The Minkowski distance of order p between two points $\mathbf{x} = (x_1, x_2, \cdots, x_n), \mathbf{y} = (y_1, y_2, \cdots, y_n) \in \mathbb{R}^n$ is defined as

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{1}$$

The following figure Fig. 2 shows unit circles (the set of all points which are at the unit distance from the center) with different values of p.

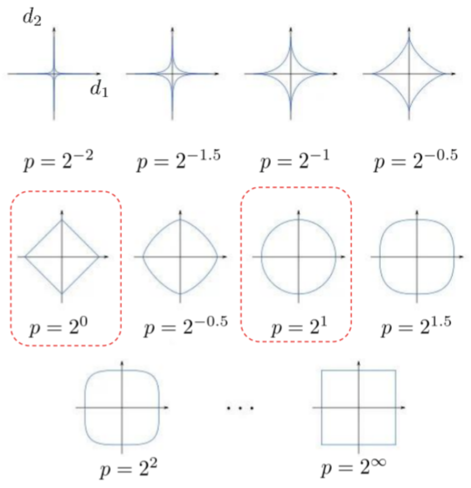


Fig. 2. Unit Circles

2) Chebyshev Distance: The Chebyshev distance or L_∞ metric is a metric defined on a vector space where the distance between two points is the maximum of their differences along any coordinate dimension. [2] It is actually the Minkowski distance when $p \to \infty$ and thus is defined as:

$$d_{Chebyshev}(\mathbf{x}, \mathbf{y}) = \lim_{p \to \infty} \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} = \max_i(|x_i - y_i|) \tag{2}$$

3) Euclidean Distance: The Euclidean distance is the "ordinary" straight-line distance in Euclidean space. It is also a specialization of Minkowski distance where $p = 2$. Thus, it is defined as

$$d_{Euclidean}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3}$$

4) Manhattan Distance: The Manhattan distance is a metric in which the distance between two points is the sum of the abosolute differences of their Cartesian coordinates. The name alludes to the island of Manhattan, which causes the shortest paths a car could take between two intersections to have the same length Fig. 3. As it's the Minkowski distance with $p = 1$, it is defined as:

$$d_{Manhattan}(\mathbf{x}, \mathbf{y}) = \left| \sum_{i=1}^{n} (x_i - y_i) \right| \tag{4}$$

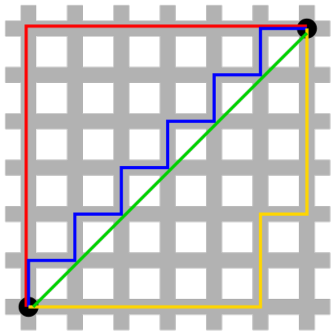


Fig. 3. Manhattan vs Euclidean

5) Cosine Distance: The Cosine distance is closely related to Cosine similarity where the more similar two points are, the closer their distance is. The Cosine similarity between two non-zero points of an inner product space is defined to equal to the cosine of the angle between them. Then, we can define it as:

$$similarity(\mathbf{x}, \mathbf{y}) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{5}$$

Thus, the Cosine distance is defined as:

$$d_{cosine}(\mathbf{x}, \mathbf{y}) = 1 - similarity(\mathbf{x}, \mathbf{y}) \tag{6}$$

6) Mahalanobis Distance: The Mahalanobis distance is a measure of the distance between a point and a distribution. It can be defined as a dissimilarity between two random vectors $\mathbf{x}$ and $\mathbf{y}$ of the same distribution with the covariance matrix $\mathbf{S}$.

$$d_{Mahalanobis}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{S}^{-1} (\mathbf{x} - \mathbf{y})} \tag{7}$$

In a metric learning task, the aim is to find the covariance matrix $\mathbf{S}$ in the Mahalanobis distance in order to generate a new distance metric. In metric learning, $\mathbf{S}^{-1}$ is always denoted as $\mathbf{M}$ and is a positive semi-definite matrix. As known, a positive semi-definite matrix $\mathbf{M}$ can be decomposed as $\mathbf{M} = \mathbf{L}^T \mathbf{L}$ for some $\mathbf{L}$. Thus, the Mahalanobis distance between $\mathbf{x}$ and $\mathbf{y}$ is now:

$$d_{Mahalanobis}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{Lx} - \mathbf{Ly})^T (\mathbf{Lx} - \mathbf{Ly})} \tag{8}$$

It is actually the Euclidean distance after a linear transformation of the feature space defined by $\mathbf{L}$. Thus, metric learning can be seen as learning a new embedding space defined by $\mathbf{L}$. It also reminds us that some learning methods who can learn linear projection matrices can be used here. In our experiments, we also tried PCA and LDA which can learn the project matrices and also reduce dimensions of the features. The equivalence of the Mahalanobis distance and the Euclidean distance after linear transformation also helped us accelerating training process. Due to the implementation of `sklearn`, any self-defined distance metric will have a slower training speed because of the heavy overhead of calling function in Python. By using linear transformation then Euclidean distance, the training time can be reduced.

C. Metric Learning

1) Information Theoretic Metric Learning: Information Theoretic Metric Learning (ITML) [3] is a kind of weakly supervised metric learning method. It minimizes the Kullback-Leibler divergence between two multivariate Gaussians subject to constraints on the associated Mahalanobis distance, which can be formulated into a Bregman optimization problem by minimizing the LogDet divergence subject to linear constraints. The multivariate Gaussian distribution associated with the Mahalanobis distance is

$$p(\mathbf{x}; \mathbf{A}) = \frac{1}{Z} \exp(-\frac{1}{2}(\mathbf{x} - \mu)^T \mathbf{A} (\mathbf{x} - \mu)) \tag{9}$$

where the inverse of Mahalanobis matrix $\mathbf{A}^{-1}$ is the covariance matrix of the Gaussian. Then, given pairs of similar points S and dissimilar points D, the learning problem is to minimize the LogDet divergence $D_{\ell \mathrm{d}}(\mathbf{A}, \mathbf{A}_0)$, which is equivalent as minimizing the K-L divergence $\mathbf{KL}(p(\mathbf{x}; \mathbf{A}_0) \| p(x(\mathbf{x}; \mathbf{A}))$:

$$\begin{array}{ccc} \min_{\mathbf{A}} D_{\ell \mathrm{d}}(\mathbf{A}, \mathbf{A}_0) = \operatorname{tr}\left(\mathbf{A}\mathbf{A}_0^{-1}\right) - \log\det\left(\mathbf{A}\mathbf{A}_0^{-1}\right) - n & & \\ \text{subject to} \quad d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) \leq u & (\mathbf{x}_i, \mathbf{x}_j) \in S \\ \qquad d_{\mathbf{A}}(\mathbf{x}_i, \mathbf{x}_j) \geq l & (\mathbf{x}_i, \mathbf{x}_j) \in D \end{array} \tag{10}$$

As we can see in Eqn. 10, u and l is the upper and lower bound of distance for similar and dissimilar pairs respectively. It means that if the Mahalanobis distance between two points is greater than l then they are considered dissimilar while if the distance is lower than u, then they are considered similar. In addition, the matrix $\mathbf{A}_0$ is the prior distance metric and in practice we always set $\mathbf{A}_0 = \mathbf{I}$ to make the prior metric the Euclidean one. So, ITML can optionally incorporate a prior on the distance function and can handle a variety of constraints. It also does not rely on eigenvalue computation or semi-definite programming.

2) Local Fisher Discriminant Analysis: Local Fisher Discriminant Analysis (LFDA) [4] is a kind of supervised learning algorithm. It is also a linear dimensionality reduction method. It is particularly useful when dealing with multi-modality, where one ore more classes consist of separate clusters in input space. The core optimization problem of LFDA is solved as a generalized eigenvalue problem. We need to define the Fisher local within-/between-class scatter matrix $\mathbf{S}^{(w)}$ and $\mathbf{S}^{(b)}$ as

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

$$\mathbf{S}^{(b)} = \frac{1}{2} \sum_{i,j=1}^{n} W_{ij}^{(b)} \left(\mathbf{x}_i - \mathbf{x}_j\right) \left(\mathbf{x}_i - \mathbf{x}_j\right)^T \tag{12}$$

where

$$W_{ij}^{(w)} = \begin{cases} 0 & y_i \neq y_j \\ \mathbf{A}_{i,j}/n_l & y_i = y_j \end{cases} \tag{13}$$

$$W_{ij}^{(b)} = \begin{cases} 1/n & y_i \neq y_j \\ \mathbf{A}_{i,j}\left(1/n - 1/n_l\right) & y_i = y_j \end{cases} \tag{14}$$

Here, $\mathbf{A}_{i,j}$ is the (i,j)-th entry of the affinity matrix $\mathbf{A}$, which can be calculated with local scaling methods. The learning problem then becomes to derive the LFDA transformation matrix $\mathbf{T}_{LFDA}$:

$$\mathbf{T}_{LFDA} = \arg\max_{\mathbf{T}} \left[\operatorname{tr}\left(\left(\mathbf{T}^T \mathbf{S}^{(w)} \mathbf{T}\right)^{-1} \mathbf{T}^T \mathbf{S}^{(b)} \mathbf{T} \right) \right] \tag{15}$$

That is, it is looking for a transformation matrix $\mathbf{T}$ such that nearby data pairs in the same class are made close and the data pairs in different classes are separated from each other; far apart data pairs in the same class are not imposed to be close.

3) Metric Learning with Application for Clustering with Side Information: Metric Learning with Application for Clustering with Side Information (MMC) [5] is also a kind of weakly supervised learning method. It minimizes the sum of squared distances between similar points, while enforcing the sum of distances between dissimilar ones to be greater than one. It is thus a convex problem and can find local-minima-free optimization solution efficiently. However, the algorithm involves the computation of eigenvalues, which is the main speed-bottleneck. According to the idea of MMC, the optimization problem is defined as

$$\begin{aligned} &\min_{\mathbf{M}\in\mathbb{S}_+^d} \textstyle\sum_{(\mathbf{x_i},\mathbf{x_j})\in S} d_{\mathbf{M}}(\mathbf{x_i},\mathbf{x_j}) \\ &\text{s.t.} \quad \textstyle\sum_{(\mathbf{x_i},\mathbf{x_j})\in D} d_{\mathbf{M}}^2(\mathbf{x_i},\mathbf{x_j}) \geq 1 \end{aligned} \tag{16}$$

where $\mathbf{M}$ is the Mahalanobis matrix and thus a positive semi-definite matrix ($\mathbf{M} \in \mathbb{S}_+^d$) and $d_{\mathbf{M}}(\cdot)$ is the Mahalanobis distance. As we can see, The algorithm aims at minimizing the sum of distances between all the similar points, while constrains the sum of distances between dissimilar points.

4) RCA: Relative Components Analysis [6] is a kind of weakly supervised learning method. It 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" which are subsets of points that are known to belong to the same class. For a training set with n training points in k chunklets (classes), the algorithm is efficient since it simply amounts to computing

$$\mathbf{C} = \frac{1}{n} \sum_{j=1}^{k} \sum_{i=1}^{n_j} (\mathbf{x}_{ji} - \mu_j)(\mathbf{x}_{ji} - \mu_j)^T \tag{17}$$

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

5) Large Margin Nearest Neighbor Metric Learning: Large Margin Nearest Neighbor Metric Learning (LMNN) [7] is a kind of supervised learning method. It learns a Mahalanobis distance metric in the k-NN classification setting. The learned metric attempts to keep close k-nearest neighbors from the same class, while keeping examples from different classes separated by a large margin. This algorithm makes no assumptions about the distribution of the data. The distance is learned by solving the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{L}} \textstyle\sum_{i,j} \eta_{ij} \left\|\mathbf{L}\left(\mathbf{x_i} - \mathbf{x_j}\right)\right\|^2 + \\ &c \textstyle\sum_{i,j,l} \eta_{ij}\left(1 - y_{ij}\right) \left[1 + \left\|\mathbf{L}\left(\mathbf{x_i} - \mathbf{x_j}\right)\right\|^2 - \left\|\mathbf{L}\left(\mathbf{x_i} - \mathbf{x_l}\right)\right\|^2\right]_+ \end{aligned} \tag{18}$$

where $\mathbf{x}_j$ is one of the k nearest neighbors sharing the same label with $\mathbf{x}_i$, $\mathbf{x}_l$ are all the other instances within that region with different labels. Both η_{ij} and y_{ij} are indicators in $\{0, 1\}$. η_{ij} represents $\mathbf{x}_j$ is the k nearest neighbors with same labels of $mathbf{x}_i$. $y_{ij} = 0$ indicates $\mathbf{x}_i, \mathbf{x}_j$ belong to different class. $[\cdot]_+ = \max(0, \cdot)$ is the Hinge loss.

II. Simple Distance Metrics

A. Minkowski Distance

In this part, we use the Euclidean distance, the Chebyshev distance and the Manhattan distance on the k-NN with our dataset. We tried different k's and the results differs with regard to different k values. The result is shown in Tab. I. As seen, the Euclidean distance performs best compared with the Chebyshev distance and the Manhattan distance under any value of k, and the Chebyshev distance performs worst. The reason that the Chebyshev distance has such poor performance must be that the Chebyshev distance between two points only takes the maximum of their differences along any coordinate dimension. Thus, the information in other dimensions is lost and the k-NN will misclassify based on incomplete information. We can also observe that the performance of the Manhattan distance is close to the one of the Euclidean distance. We deem that it's because the distances between our data points is small and the Manhattan distance (1-order distance) and the Euclidean distance (2-order distance) on the same pair of points is close.

As the k increases, the performance of k-NN first increases and then decreases. When k is small, the classifier worked not so good. It can classify most of points into right class because the points of same class always cluster together. However, the ability to expel the outliers is weak in this case, so as k increases, the performance will get better. When k is large, the performance is quite poor, it is because that in this case, the local property is lost, too much noise is added and the classifier performs bad.

We will consider the accuracy of the Euclidean distance when $k = 7$ as the baseline of later comparisons between different distance metrics.

B. Dimension Reduction and Minkowski Distance

In this part, we explored the effect of dimension reduction on k-NN with different Minkowski distances. We did experiments on different methods including PCA and LDA, different kernels

TABLE I
COMPARISON OF $k-$NN CLASSIFICATION TASK WITH DIFFERENT MINKOWSKI DISTANCES

Acc. M	$k-$NN with different Minkowski distances		
k	Euclidean(%)	Chebyshev(%)	Manhattan(%)
2	84.56	63.45	84.75
3	86.89	66.64	**86.62**
4	86.94	67.77	86.51
5	87.44	68.44	86.52
6	87.25	68.95	86.40
7	**87.61**	69.07	86.60
8	87.37	69.37	86.38
9	87.54	69.35	86.38
10	87.32	69.32	86.36
11	87.35	69.37	86.28
12	87.19	69.38	86.15
13	87.16	**69.44**	85.97
14	87.09	69.28	85.87
15	86.10	69.31	85.79
16	86.84	69.16	85.54
50	84.46	65.99	82.44
100	82.06	62.61	79.36
200	78.14	58.50	74.30
500	70.53	51.28	64.93
1000	62.51	44.21	54.95

TABLE II
COMPARISON OF LINEAR PCA+$k-$NN CLASSIFICATION TASK WITH DIFFERENT MINKOWSKI DISTANCES

Acc. M	Linear PCA+$k-$NN with different Minkowski distances		
k	Euclidean(%)	Chebyshev(%)	Manhattan(%)
2	83.96	77.00	83.58
3	86.34	79.54	85.91
4	86.70	80.40	86.42
5	87.17	80.88	87.03
6	87.18	80.90	86.80
7	87.41	81.24	87.02
8	87.50	81.07	86.88
9	87.45	81.25	87.07
10	87.33	81.24	**87.16**
11	**87.51**	**81.29**	87.13
12	87.39	81.06	87.03
13	87.50	81.13	87.03
14	87.26	80.95	86.92
15	87.23	80.97	86.94
16	86.92	80.99	86.83
50	85.20	78.92	84.87
100	83.21	76.25	83.06
200	80.35	72.71	80.48
500	75.60	67.08	75.51
1000	70.13	60.73	70.18
Baseline	**87.61**		

of PCA and different dimensions. Firstly, we tried different kernels (linear, rbf) of PCA and reduced the original 2048 dimensions into 50 dimensions. Tab. II shows the result in this part. The tendency of the performance when k increases and the differences of performance among different distance metrics keep the same as the case before (without dimension reduction). What's surprising is that even though the dimension is reduced into 50 (40 times smaller than before), the performance has slightly decreased. The best performance of the Euclidean distance when not conducting dimension reduction is 87.61% and 87.51% after dimension reduction. Such result reminds us of using dimension reduced data to train our metric learners later to avoid memory explosion and speed up training process.

We then conducted experiments on PCA with RBF kernel. Comparing Tab. II and Tab. III We found that is performed slightly worse than Linear PCA, but still well. For this dataset and k-NN classification task, Linear maybe a suitable kernel. The RBF kernel may find a different low dimension embedding space of the dataset and thus performed different from the Linear kernel.

In Tab. IV, we also compared the performance of k-NN with the Euclidean distance on dimension reduced dataset of different dimensions. The performance when we reduced the dataset into 500 dimensions is surprisingly poor compared to the original dataset and the dataset whose dimension os reduced to 50. Generally, as the dimension decreases, some information may get lost and the performance of the classifier will get worse. However, though the performance decreased when dimension is 500 compared with original dataset. It increased when dimension is 50 compared with the dataset in dimension 500. We think that when reducing the dimension of the original dataset to 500 dimensions, the local structure or the cluster is broken by PCA. In the new embedding space,

TABLE III
COMPARISON OF RBF PCA+$k-$NN CLASSIFICATION TASK WITH DIFFERENT MINKOWSKI DISTANCES

Acc. M	RBF PCA+$k-$NN with different Minkowski distances		
k	Euclidean(%)	Chebyshev(%)	Manhattan(%)
2	82.72	76.65	82.89
3	85.45	79.34	85.71
4	85.84	80.05	85.95
5	86.29	80.76	86.49
6	86.32	80.92	86.53
7	**86.74**	81.28	86.75
8	86.60	81.31	86.70
9	86.57	81.37	**86.84**
10	86.56	81.47	86.73
11	86.47	81.39	86.78
12	86.38	81.41	86.64
13	86.40	81.55	86.68
14	86.48	81.64	86.64
15	86.36	81.59	86.55
16	86.49	**81.67**	86.63
50	84.53	79.49	84.57
100	82.61	77.42	83.18
200	80.16	74.69	80.49
500	75.54	69.49	75.66
1000	69.84	64.02	70.23
Baseline	**87.61**		

the data points with different labels mixed together and there is less clusters of same class than before, which leads to a bad performance. These are just our guesses, the reason may be explored by further experiments. However, the results reminds us of choosing a dimension to be reduced to carefully.

In the last project, we introduced another feature projection algorithm to reduce dimension which is Linear Discriminate Analysis (LDA). Its main idea is to project the samples on a straight line so that the projections of similar samples are as close as possible, and the projection points of heterogeneous samples are as far as possible. This property will improve the performance of k-NN a lot, because it increases the data locality

TABLE IV

COMPARISON OF $k-$NN CLASSIFICATION TASK WITH EUCLIDEAN DISTANCE IN DIFFERENT DIMENSIONS

Acc. M / k	$k-$NN with Euclidean distance in different dimensions		
	Linear,50(%)	Linear,500(%)	Original,2048(%)
2	83.96	**66.01**	84.56
3	86.34	64.72	86.89
4	86.70	63.65	86.94
5	87.17	61.72	87.44
6	87.18	60.39	87.25
7	87.41	59.01	**87.61**
8	87.50	57.72	87.37
9	87.45	56.57	87.54
10	87.33	55.44	87.32
11	**87.51**	54.30	87.35
12	87.39	53.45	87.19
13	87.50	52.20	87.16
14	87.26	51.55	87.09
15	87.23	50.55	86.10
16	86.92	49.45	86.84
50	85.20	32.90	84.46
100	83.21	22.63	82.06
200	80.35	13.87	78.14
500	75.60	06.67	70.53
1000	70.13	05.17	62.51
Baseline	**87.61**		

making points of same label closer and draws outliers far away. The comparison between LDA and PCA in Tab. V shows that LDA must be a better dimension reduction method and it even improved the performance compared with the performance on non-dimension-reduced dataset. We first reduced the dimension into 40 and feed the data after reduction into k-NN with the Euclidean distance. In any settings of k, the LDA's performance exceeds the baseline of 87.61%. We will also consider LDA as a preprocessing method in metric learning tasks. In the later experiments, we will focus on k-NN with k in range of $[2, 3, \cdots, 15, 16]$ because when k is large, the performance must be poor.

In general, PCA and LDA learns a projection matrix to project original dataset into a new embedding space. According to our introduction of the Mahalanobis distance, these two methods can also be regarded as metric learning methods, but here, we preferred to regard them as preprocessing methods.

C. Cosine Distance

We only take a glimpse at the cosine distance due to the time limit. The training and testing of k-NN with self-defined distance metric (a function as a parameter of k-NN classifier) is much more slower than using other distance metrics (Euclidean and so on) in `sklearn`. An experiment under a setting of parameters will consume a day or more time to train and test. Let's compare k-NN with the Euclidean distance and the Cosine distance on PCA preprocessed dataset in range $[2, 3, \cdots, 12]$ of k. In Tab. VI, we can also observe that the Cosine distance will perform better than the Euclidean distance and will sometimes exceeds the baseline. It shows that the Cosine distance can be a good distance metric in k-NN.

TABLE V

COMPARISON OF $k-$NN CLASSIFICATION TASK WITH LINEAR PCA AND LDA

Acc. M / k	$k-$NN with Linear PCA and LDA	
	PCA,50(%)	LDA,40(%)
2	83.96	88.21
3	86.34	89.95
4	86.70	89.91
5	87.17	90.41
6	87.18	90.26
7	87.41	90.46
8	87.50	90.55
9	87.45	90.62
10	87.33	90.58
11	**87.51**	90.65
12	87.39	90.70
13	87.50	90.76
14	87.26	90.68
15	87.23	**90.77**
16	86.92	90.74
Baseline	**87.61**	

TABLE VI

COMPARISON OF PCA+$k-$NN CLASSIFICATION TASK WITH EUCLIDEAN DISTANCE AND COSINE DISTANCE

Acc. M / k	PCA+$k-$NN with Euclidean distance and Cosine distance	
	Euclidean(%)	Cosine(%)
2	83.96	84.49
3	86.34	86.90
4	86.70	87.07
5	87.17	87.69
6	87.18	87.69
7	87.41	**88.25**
8	87.50	87.94
9	87.45	88.13
10	87.33	88.07
11	**87.51**	88.03
12	87.39	88.09
Baseline	**87.61**	

III. METRIC LEARNING

A. Information Theoretic Metric Learning (ITML)

In this experiment, we trained supervised ITML model in training dataset, then transform source data to ITML-learned data for KNN classification. We set $constraints = 200$ in supervised ITML model and give the $k-$neighbors range in $k-$NN model as $[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$. For PCA-preprocessed ITML, we first perprocess the data by 50-conponents linear PCA, then train supervised ITML model on the processed data, it will give a better performance. For LDA-preprocessed one, we perprocess the data by 40-conponents LDA, instead.

Our experiment results are shown in Tab. VII. Compared with baseline 87.61% in Euclidean KNN, we can find that simple ITML model and PCA-preprocessed ITML model have poor performance, we deem the reason is that ITML minimize the LogDet divergence subject to linear constraints while it does not rely on an eigenvalue computation, and PCA can't differ the classes which is useless for $k-$NN classification. But we can find PCA-preprocessed model have better performance than pure ITML model, we think PCA method can have compensation on eigenvalue computation for ITML. When

TABLE VII
COMPARISON OF VARIANCE ITML AND BASELINES IN $k-$NN CLASSIFICATION TASK

Acc. M / k	Variance ITML model + $k-$NN		
	ITML(%)	PCA-preprocessed(%)	LDA-preprocessed(%)
2	82.04	83.18	88.06
3	**83.60**	85.41	89.57
4	83.31	85.55	89.42
5	83.52	86.19	90.01
6	83.33	86.21	90.04
7	83.37	**86.67**	90.35
8	83.02	86.51	90.35
9	82.97	86.37	90.40
10	82.72	86.41	90.55
11	82.73	86.25	90.64
12	82.52	86.27	90.67
13	82.47	86.17	90.58
14	82.33	86.06	90.64
15	82.10	86.15	**90.74**
16	81.96	86.01	90.69
Baseline	**87.61**		

we set $k = 15$, we have best performance 90.74%, we think LDA method can make the variance high enough between different clusters.

B. Local Fisher Discriminant Analysis (LFDA)

In this experiment, we trained LFDA model in training dataset, then project source data to LFDA-learned data for KNN classification. We set $k-$neighbors range in $k-$NN model as $[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$. For PCA-preprocessed LFDA, we first perprocess the data by 50-conponents linear PCA, then train supervised ITML model on the processed data, it will give a better performance. For LDA-preprocessed one, we perprocess the data by 40-conponents LDA, instead.

TABLE VIII
COMPARISON OF VARIANCE LFDA AND BASELINES IN $k-$NN CLASSIFICATION TASK

Acc. M / k	Variance LFDA model + $k-$NN	
	PCA-preprocessed(%)	LDA-preprocessed(%)
2	84.00	88.18
3	86.30	89.95
4	86.74	89.94
5	87.17	90.38
6	87.16	90.29
7	87.40	90.43
8	87.42	90.54
9	87.51	90.62
10	87.46	90.55
11	**87.51**	90.63
12	87.38	90.68
13	87.49	90.71
14	87.24	90.68
15	87.23	**90.76**
16	86.97	90.76
Baseline	**87.61**	

Our experiment results are shown in Tab. VIII. We find simple LFDA model have poor performance as 16.20%, we think this may be caused by that we set the n_components parameter as default, and it means we can't control its dimensionality reduction process to get a satisfied result. Compared with baseline 87.61% in Euclidean KNN, we find LDA-preprocessed model reached 90.76% while PCA-preprocessed one reached 87.51%, we deem the reason is that LDA can make the variance high enough between different clusters.

C. Mahalanobis Metric Learning for Clustering (MMC)

In this experiment, we trained MMC model in training dataset, then project source data to MMC-learned data for KNN classification. We set $k-$neighbors range in $k-$NN model as $[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$. For PCA-preprocessed LFDA, we first perprocess the data by 50-conponents linear PCA, then train MMC model on the processed data, it will give a better performance. For LDA-preprocessed one, we perprocess the data by 40-conponents LDA, instead.

TABLE IX
COMPARISON OF VARIANCE MMC AND BASELINES IN $k-$NN CLASSIFICATION TASK

Acc. M / k	Variance MMC model + $k-$NN		
	MMC(%)	PCA-preprocessed(%)	LDA-preprocessed(%)
2	84.89	83.73	87.39
3	87.29	86.48	89.38
4	87.38	86.35	89.47
5	87.66	87.02	89.80
6	87.58	86.79	89.91
7	**87.82**	87.22	90.05
8	87.72	87.01	90.04
9	87.46	**97.23**	90.16
10	87.46	87.12	90.07
11	87.53	87.09	90.24
12	87.48	87.05	90.24
13	87.60	86.95	90.26
14	87.44	87.02	90.30
15	87.49	86.99	**90.34**
16	87.27	86.97	90.28
Baseline	**87.61**		

Our experiment results are shown in Tab. IX. We find PCA-preprocessed MMC reached 97.23%, while simple MMC reached 87.82% and LDA-preprocessed MMC reached 90.34%. We speculate that this is due to the reduction of overfitting after dimensionality reduction. In addition, the idea of LDA is slightly similar to that of MMC, both of which are to close the data of the same class and push away the data of different classes. It may be that the data has been processed twice with similar effects, resulting in experimental performance that is not as good as PCA-preprocessed MMC.

D. Relevant Component Analysis (RCA)

In this experiment, we trained MMC model in preprocessed training dataset for lacking of computing resourses, then project source data to MMC-learned data for KNN classification. We set $k-$neighbors range in $k-$NN model as $[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$. For PCA-preprocessed RCA, we first perprocess the data by 50-conponents linear PCA, then train RCA model on the processed data. For LDA-preprocessed one, we perprocess the data by 40-conponents LDA, instead.

Our experiment results are shown in Tab. X. We find LDA-preprocessed RCA which reached 90.34% performed better

TABLE X
COMPARISON OF VARIANCE RCA AND BASELINES IN $k-$NN CLASSIFICATION TASK

Acc. M	Variance RCA model + $k-$NN	
k	PCA-preprocessed(%)	LDA-preprocessed(%)
2	82.65	87.39
3	85.15	89.38
4	85.37	89.47
5	85.83	89.80
6	86.07	89.91
7	**86.30**	90.05
8	86.14	90.04
9	86.12	90.16
10	85.96	90.07
11	86.03	90.24
12	86.09	90.24
13	86.09	90.26
14	85.90	90.30
15	85.83	**90.34**
16	85.73	90.28
Baseline	**87.61**	

than PCA-preprocessed RCA which reached 86.30%. RCA uses the information in such chunklets to reduce irrelevant variability in the data while amplifying relevant variability. Therefore, we deem the reason is that PCA lacks attention to categories when reducing dimensionality.

E. Large Margin Nearest Neighbor (LMNN)

In this experiment, we trained LMNN model in PCA-preprocessed training dataset and LDA-preprocessed training dataset, then project the data to LMNN-learned data for KNN classification. Due to lack of computing resources, we did not use the original dataset. LMNN needs to set the target neighbor k, so we also set different target neighbors for experiment. We set $K-$neighbors in KNN range as $[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$. For PCA-preprocessed LFDA, we set $k-$target neighbors in LMNN range as $[2, 3, 4, 5, 6]$. We first perprocess the data by 50-conponents linear PCA, then train LMNN model on the processed data. For LDA-preprocessed one, we perprocess the data by 40-conponents LDA instead and set $k-$target neighbors in LMNN range as $[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$.

Our experiment results are shown in Tab. XI, Tab. XII, Tab. XIII, Tab. XIV. We find both PCA-processed LMNN and LDA-processed LMNN have improved the performance of KNN. We think the main reason is that LMNN reduces the distance between similar samples and expands the distance between different samples. Compared with PCA-preprocessed LMNN which reached 88.24%, LDA-preprocessed LMNN performed better which reached 90.99%. We deem the reasion is that the idea of LDA is to minimize the intra-class variance and maximum inter-class variance after projection, which is helpful for classification. But we only set the $k-$target neighbors in LMNN change between 2-6 for PCA-preprocessed LMNN, this factor may also affect the experimental results. Another interesting finding is that the best experimental result is not achieved when $k = K$. We speculate that $k-$target neighbors represent our degree of supervision of the model. But we must

TABLE XI
COMPARISON OF VARIANCE PCA+LMNN AND BASELINES IN $k-$NN CLASSIFICATION TASK

Acc. M	Variance PCA+LMNN model + $k-$NN				
k	k=2(%)	k=3(%)	k=4(%)	k=5(%)	k=6(%)
2	84.65	84.66	84.55	84.79	84.84
3	87.01	87.19	86.98	87.05	87.11
4	87.37	87.42	87.48	87.24	87.27
5	87.80	87.83	87.87	87.98	87.70
6	87.73	87.80	87.76	87.90	87.82
7	88.07	88.05	88.09	**88.21**	88.19
8	87.99	87.95	88.05	88.09	88.03
9	88.18	**88.13**	88.21	88.16	88.19
10	87.99	88.06	88.10	88.05	88.19
11	**88.18**	88.04	88.12	88.18	88.13
12	88.07	88.01	88.07	88.02	88.13
13	88.03	88.03	**88.24**	88.15	88.21
14	97.90	97.90	87.96	88.09	88.19
15	87.90	88.00	88.10	88.10	**88.23**
16	87.80	87.90	88.05	88.03	88.21
Baseline	**87.61**				

TABLE XII
COMPARISON OF VARIANCE LDA+LMNN AND BASELINES IN $k-$NN CLASSIFICATION TASK

Acc. M	Variance LDA+LMNN model + $k-$NN					
k	k=2(%)	k=3(%)	k=4(%)	k=5(%)	k=6(%)	k=7(%)
2	88.37	88.52	88.33	88.37	88.29	88.41
3	89.84	90.12	90.06	89.92	89.92	89.83
4	89.79	89.95	90.05	90.07	90.03	89.93
5	90.37	90.46	90.54	90.50	90.41	90.44
6	90.07	90.44	90.33	90.20	90.27	90.30
7	90.46	90.62	90.68	90.73	90.72	90.68
8	90.53	90.64	90.58	90.58	90.68	90.59
9	90.61	90.74	90.66	90.72	90.82	90.73
10	90.56	90.71	90.68	90.72	90.69	90.66
11	**90.77**	90.77	**90.82**	90.82	90.91	90.85
12	90.64	90.70	90.75	90.76	90.78	90.87
13	90.77	**90.87**	90.80	90.83	90.82	**90.89**
14	90.72	90.82	90.77	90.80	90.76	90.85
15	90.71	90.84	90.73	90.83	**90.99**	90.81
16	90.71	90.86	90.82	**90.88**	90.94	90.86
Baseline	**87.61**					

TABLE XIII
COMPARISON OF VARIANCE LDA+LMNN AND BASELINES IN $k-$NN CLASSIFICATION TASK (CONTINUE)

Acc. M	Variance LDA+LMNN model + $k-$NN					
k	k=8(%)	k=9(%)	k=10(%)	k=11(%)	k=12(%)	k=13(%)
2	88.31	88.26	88.30	88.21	88.29	88.33
3	89.87	90.02	89.92	89.91	90.00	89.96
4	90.02	90.09	90.06	89.99	89.97	89.97
5	90.54	90.48	90.58	90.46	90.48	90.43
6	90.34	90.36	90.35	90.29	90.32	90.37
7	90.70	90.60	90.64	90.61	90.60	90.61
8	90.60	90.50	90.48	90.48	90.57	90.54
9	90.77	90.76	90.70	90.74	90.64	90.83
10	90.63	90.63	90.69	90.61	90.67	90.65
11	90.74	90.78	90.78	90.75	90.82	90.81
12	90.86	90.82	90.80	90.78	90.80	90.79
13	**90.87**	90.85	90.86	**90.84**	90.90	90.83
14	90.87	**90.93**	**90.93**	90.82	90.82	90.83
15	90.86	90.88	90.89	90.84	**90.91**	90.89
16	90.86	90.88	90.93	90.78	90.89	**90.91**
Baseline	**87.61**					

TABLE XIV
COMPARISON OF VARIANCE LDA+LMNN AND BASELINES IN $k-$NN CLASSIFICATION TASK (CONTINUE)

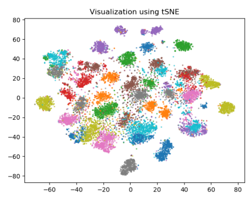
Acc. M / k	Variance LDA+LMNN model + $k-$NN		
	k=14(%)	k=15(%)	k=16(%)
2	88.30	88.31	88.21
3	90.01	90.06	89.92
4	90.07	90.09	89.97
5	90.42	90.39	90.43
6	90.28	90.23	90.32
7	90.54	90.54	90.51
8	90.54	90.50	90.56
9	90.70	90.68	90.66
10	90.68	90.67	90.69
11	90.78	90.72	90.69
12	90.76	90.79	90.69
13	90.78	90.85	90.76
14	90.79	90.82	90.82
15	90.83	**90.97**	90.83
16	**90.87**	90.93	**90.89**
Baseline	**87.61**		

also consider the amount of data for each class. Bcause when the number of certain types of data is less than k, there is not enough correct target neighbors to calculate the loss function, which will have a bad impact on the results. Therefore, the relationship between k and K for best results involves the trade-off between the data set itself and the intensity of supervision. We believe that if the parameter of $k-$target neighbors in LMNN can change according to each type in the dataset instead of the initial setting, it will play an optimizing role for LMNN.

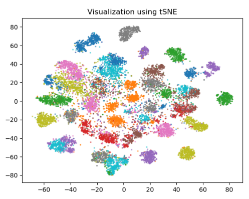
IV. CONCLUSION

In this project, we introduced traditional distance metric and several metric learning method, which is to learn a distance metric for the input space of data from a given collection of pair of similar/dissimilar points that preserves the distance relation among the training data. We have several simple distance matrics experiments like Minkowski, Chebyshev, Euclidean, Manhattan, Cosine and Mahalanobis, besides, we have several supervised experiments such as ITML, LFDA, MMC, RCA and LMNN, and give a experiment with unsupervised method PCA. In this section, we will conclude the pros and cons of every method and evaluate the performance of simple distance metrics and metrics learning models.

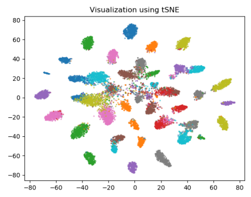
For simple metrics, we can find different methods should be applied to different data structures, which should be discussed in detail. Metrics learning methods can combined with simple metrics and usually get better results. For metrics learning methods, we find if preprocessed method is not used, LFDA's result is not satisfied, we deem the reason that the $n\_compoents$ parameter in FLDA is not set but use default one. In order to save the memory consumption of training model in RCA and LMNN, preprocess method must be attached, and another reason is that with PCA-preprocessed method or LDA-preprocessed method, we can get better result. In Tab. XV, we show performance comprision of all the metrics learning methods and we will analyse it as follows.



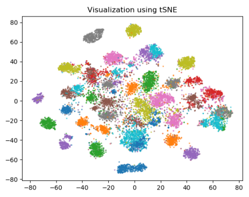
(a) 2D-scattering



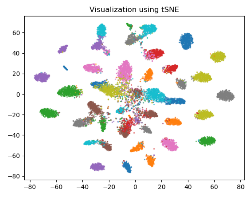
(b) PCA 2D-scattering



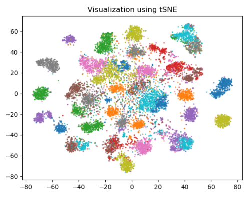
(c) LDA 2D-scattering



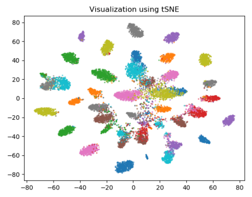
(d) PCA + LMNN 2D-scattering



(e) LDA + LMNN 2D-scattering



(f) PCA + MMC 2D-scattering



(g) LDA + MMC 2D-scattering

Fig. 4. tSNE Visualization with Variance Model

As is shown in Fig. 4, we can find LDA can maintain the distance better between different cluster than PCA, it can explain why LDA-preprocessed method usually have better performance than PCA-preprocessed one, for instance, (d)(e) can subscribe the reason why $k-$NN with LDA-preprocessed method achieve higher accuracy in LMNN. (f)(g) may contradict the experimental results that PCA-preprocessed MMC have better performance than LDA-preprocessed one, we deem the reason that when we set $n\_components = 50$ in PCA, it performs well but tSNE can't show this property.

REFERENCES

[1] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[2] J. M. Abello, P. M. Pardalos, and M. G. C. Resende, *Handbook of Massive Data Sets*. Springer, 2002.

[3] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon, "Information-theoretic metric learning," in *Machine Learning, Twenty-fourth International Conference, Corvallis, Oregon, Usa, June*, 2007.

TABLE XV
COMPARISON OF METRICS LEARNING AND BASELINES IN $k-$NN CLASSIFICATION TASK

preM+Acc. / M	Variance metrics learning model + $k-$NN	
	preprocess model(%)	best performance(%)
ITML	LDA	90.74
LFDA	LDA	90.76
MMC	PCA	87.23
RCA	LDA	90.34
LMNN	LDA	90.99
Baseline	**87.61**	

[4] M. Sugiyama, "Dimensionality reduction of multimodal labeled data by local fisher discriminant analysis," *Journal of machine learning research*, vol. 8, no. May, pp. 1027–1061, 2007.

[5] E. P. Xing, M. I. Jordan, S. J. Russell, and A. Y. Ng, "Distance metric learning with application to clustering with side-information," in *Advances in neural information processing systems*, 2003, pp. 521–528.

[6] N. Shental, T. Hertz, D. Weinshall, and M. Pavel, "Adjustment learning and relevant component analysis," in *European conference on computer vision*. Springer, 2002, pp. 776–790.

[7] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," *Journal of Machine Learning Research*, vol. 10, no. Feb, pp. 207–244, 2009.