DISTANCE METRIC

Dinghua Zhao
Department of Computer Science
Shanghai Jiao Tong University
Minhang District, Shanghai, PRC
zhaodinghua@sjtu.edu.cn

Hanzhang Yang
Department of Computer Science
Shang Hai Jiaotong University
Minhang District, Shanghai, PRC
linqinluli@sjtu.edu.cn

Qianfei Ren
Department of Computer Science
Shang Hai Jiaotong University
Minhang District, Shanghai, PRC
cordial.Daniel@sjtu.edu.cn

Abstract

Distance measurement is widely used in classical machine learning methods. The distance between samples or distributions is used to measure the similarity between them.Metric learning algorithms can be classified from the perspectives of dimensionality reduction, nearest neighbors classifier, cluster centers and information theory.In our experiment, we implement a KNN model to complete the classification work on a given dataset, and apply a series of different distance measures to explore the performance of KNN classification work, and do some visualization work. Then, we implement some classic simple distance metrics and compare their performance. Finally, we explore a series of measurement learning methods based on time cost, space cost and accuracy, and analyze their different performance.

1 Introduction

In our experiment, we choose the awa2 dataset provided by our teacher as our dataset, which contains 37322 pictures and 2048-d features as the original input, and mark the pictures as 50 classes as the output. We classify 60% data as training set and 40% data as test set, and construct a KNN model. In this paper, we mainly introduce KNN in the second part. In the third part, we mainly introduce and test different distance metric learning methods. Finally, we will compare and analyze the performance of different distance metric learning methods.

Our lab based on Python 3.7, and requires these library:scikit-learn, numpy, metric-learn

2 Method Analysis

2.1 K-Nearest Neighbor(KNN)

The core idea of the KNN algorithm is that if most of the K nearest samples in the feature space of a sample belong to a certain category, the sample also belongs to this category and has the characteristics of the samples in this category. In determining the classification decision, this method only determines the category of the sample to be classified based on the category of the nearest one or several samples. The KNN method is only related to a very small number of adjacent samples when making category decisions.

The KNN algorithm needs to calculate the distance between the test sample point (point to be classified) and each other sample point. In order to discuss the influence of the distance metric on the KNN algorithm, this experiment is based on KNN.

2.2 Simple Distance Metrics

KNN needs to compute the distance between pairs of datapoints. In this section, distance is computed by a determin-istic function.

2.2.1 Minkowski Distance

The distance of two vectors $\{x_1, x_2, \cdots, x_n\}$ and $\{y_1, y_2, \cdots, y_n\}$ is defined as this:

$$d(x, y) = (\sum_i |x_i - y_i|^p)^{\frac{1}{p}}$$

For different p values, we can generate different distance metrix.

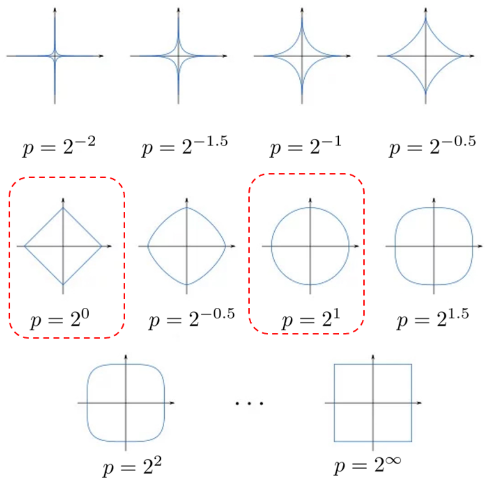


Figure 1: Different P Value

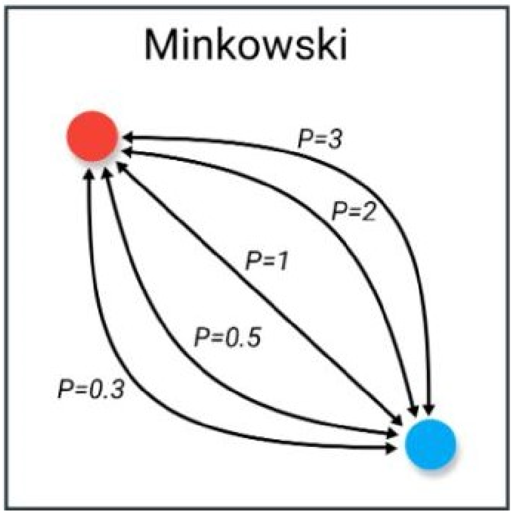


Figure 2: Different P Value

1. $p = 1$, Manhattan Distance

$$d(X, Y) = \sum_i |x_i - y_i|$$

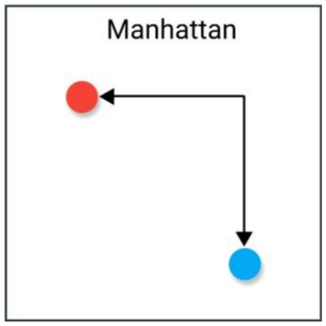


Figure 3: Manhattan Distance

2. $p = 2$, Euclidean Distance

$$d(X, Y) = \sqrt{\sum_i (x_i - y_i)^2}$$

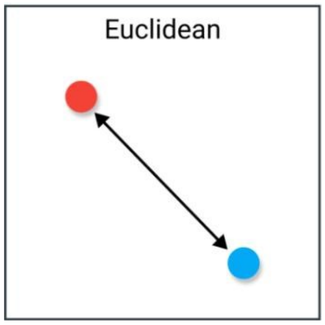


Figure 4: Euclidean Distance

3. $p = \inf$. Chebyshev Distance

$$d(X, Y) = \max_i |x_i - y_i|$$

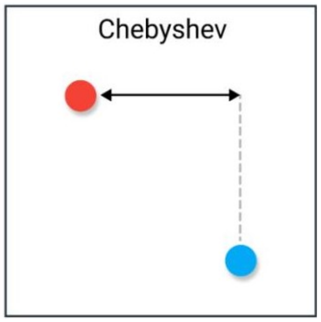


Figure 5: Chebyshev Distance

2.2.2 Cosine Distance

Cosine similarity measures similarity of two data points by the angle between them:

$$\cos X, Y = \frac{X^T Y}{\|X - Y\|}$$

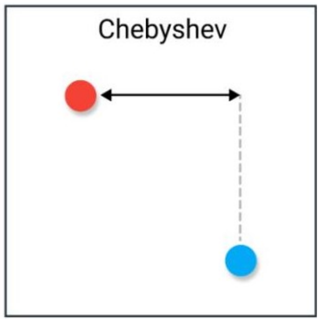


Figure 6: Euclidean Distance

To scale distance to [0, 1] and transform similarity to distance, cosine distance is defined as

$$d(X,Y) = \frac{1}{2}(1 - \cos(X,Y))$$

2.3 Metric Learning

2.3.1 Mahalanobis Metric Learning for Clustering (MMC)

MMC minimizes the sum of squared distances between similar points, while enforcing the sum of distances between dissimilar ones to be greater than one. 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. However, it is one of the earliest and a still often cited technique.

The algorithm aims at minimizing the sum of distances between all the similar points, while constrains the sum of distances between dissimilar points:

$$\min_{M \in S} \sum_{(X_i, X_j) \in S} d_M(X_i, X_j)$$

$$\sum_{(X_i, X_j) \in D} d_M^2(X_i, X_j) \geq 1$$

2.3.2 Relative Components Analysis (RCA)

RCA 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

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

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

2.3.3 Large Margin Nearest Neighbor (LMNN)

The full name of lmnn is large margin nearest neighbors. The purpose of lmnn is to learn a low dimensional mapping to make the algorithm based on k-nearest neighbors perform best. First of all, we will introduce two concepts: target neighbors and imposter.For sample x_i, select the nearest K samples of the same class. These samples are called the target neighbors of X_i. if x_i is one of the target neighbors, it is represented by $j \overset{i}{\rightarrow}$; If there is a different class of sample x_l satisfying $||x_i - x_l||^2 <= ||x_i - x_j||^2 + 1$, then the sample is called impaster of X_i, x_j.

Lmnn algorithm includes two forces, namely pull and push. First, we introduce pull to bring target neighbors closer

$$\epsilon_{pull} = \sum_i \sum_{j \rightarrow i} ||L(x_j - x_i)||^2$$

Push pushes the imposter further

$$\epsilon_{push} = \sum_i \sum_{j \to i} \sum_l (1 - y_{il}) \left[1 + \|L(x_i - x_j)\|^2 - \|L(x_i - x_l)\|^2\right]_+$$

Then the final loss includes two parts, which are weighted

$$\epsilon(L) = (1 - \mu)\epsilon(pull) + \mu\epsilon(push)$$

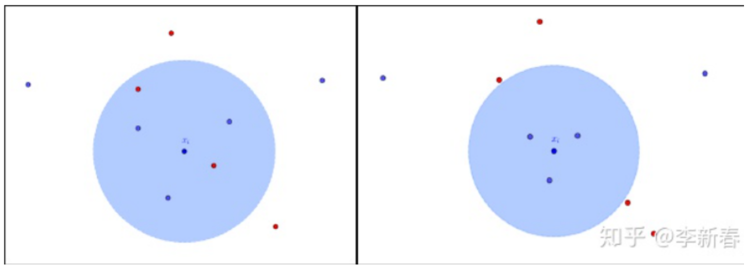
Of course, you can also replace d_l with d_m.

$$\epsilon(M) = (1 - \mu) \sum_i \sum_{j \to i} \|x_j - x_i\|_M^2$$
$$+ \mu \sum_i \sum_{j \to i} \sum_l (1 - y_{il}) \left[1 + \|x_i - x_j\|_M^2 - \|x_i - x_l\|_M^2\right]_+$$

At this time, the problem becomes a convex optimization problem, which can be solved by using the method of subgradient descent and projection into the positive semi definite matrix space.

The sub gradient is as follows

$$G = (1 - \mu) \sum_i \sum_{j \to i} O_{ij} + \mu \sum_{i,j,l \in \text{imposters}} (O_{ij} - O_{il}),$$
$$O_{ij} = (x_i - x_j)(x_i - x_j)^T$$



2.3.4 Neighborhood Components Analysis (NCA)

NCA is a distance metric learning algorithm which aims to improve the accuracy of nearest neighbors classification compared to the standard Euclidean distance. The algorithm directly maximizes a stochastic variant of the leave-one-out k-nearest neighbors (KNN) score on the training set. It can also learn a low-dimensional linear transformation of data that can be used for data visualization and fast classification.

They use the decomposition $M = L^T L$ and define the probability $p_i j$ that x_i is the neighbor of x_j by calculating the softmax likelihood of the Mahalanobis distance:

$$p_{ij} = \frac{exp(-\|L_{x_i} - L_{x_j}\|_2^2)}{\sum exp(-\|L_{x_i} - L_{x_j}\|_2^2)}$$

Then the probability that x_i will be correctly classified by the stochastic nearest neighbors rule is:

$$p_i = \sum_{j:y_i = y_j} p_{ij}$$

The optimization problem is to find matrix L that maximizes the sum of probability of being correctly classified:

$$L = argmax \sum_i p_i$$

2.3.5 Local Fisher Discriminant Analysis (LFDA)

LFDA is a kind of supervised learningalgorithm. 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 $S^{(w)}$ and $S^{(b)}$ as:

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

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

from where we have:

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

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

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

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

That is, it is looking for a transformation matrix 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.

2.3.6 ITML

The full name of itml is information theoretical metric learning. Itml can be described as a Bregman optimization problem by minimizing the relative entropy of two multigaussian distributions with Mahalanobis distance constraints, also known as Kullback Leibler divergence (KL divergence), and by minimizing the logdet divergence with linear constraints. The algorithm can deal with all kinds of constraints, and can choose to add a priori distance function. Unlike other methods, itml does not rely on eigenvalue calculation or semidefinite programming. Let's first look at the relationship between Gaussian distribution and distance measure

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2} \det(\Sigma)^{1/2}} \exp\left((x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

The definition of Mahalanobis distance is as follows

$$d_M^2(x, \mu) = (x - \mu)^T M (x - \mu)$$

Therefore, given a M, we can get a Mahalanobis distance and a Gaussian distribution

$$p(x|M) = \frac{1}{(2\pi)^{d/2} \det(M)^{1/2}} \exp\big((x - \mu)^T M^{-1} (x - \mu)\big)$$

Then, if we have a priori M_0, what we need to do is to minimize the KL distance between the two distributions

$$\begin{aligned} &\min_{M \in S_d(\mathcal{R})^+} KL(p(x|M_0)||P(x|M)) \\ s.t. \quad & d_M(x_i, x_j) \le u, \quad (i, j) \in S \\ & d_M(x_i, x_k) \ge v, \quad (i, j) \in D \end{aligned}$$

Moreover, according to the results of the previous two KL Gaussian distributions, the expression of log det divergence can be obtained

$$\begin{aligned} &\min_{M \in S_d(\mathcal{R})^+} D_{ld}(M_0, M) \\ s.t. \quad & d_M(x_i, x_j) \le u, \quad (i, j) \in S \\ & d_M(x_i, x_k) \ge v, \quad (i, j) \in D \end{aligned}$$

2.3.7 Metric Learning from Relative Comparisons by Minimizing Squared Residual (LSML)

Lsml proposes a simple and effective algorithm to minimize the convex objective function corresponding to the sum of squares of the constrained residuals. The algorithm adopts the constraint form of relative distance comparison, which is especially suitable for the situation that pairwise constraints cannot be obtained naturally, making it difficult to deploy the algorithm based on pairwise constraints. In addition, when the dimension is large and only a few constraints are given, its sparsity can make the estimation more stable. Each constraint $d(x_a, x_b)$ in the loss function is expressed in the following form

$$H(d_M(X_a, X_b) - d_M(X_c, X_d))$$

Where H (.) is the square hinge loss function

$$H(x) = \begin{cases} 0 & x \le 0 \\ x^2 & x > 0 \end{cases}$$

3 Experiment and Result Analysis

3.1 Dataset and Pro-processing

The data set of this experiment still uses the AwA2 data set with pre-extracted deep learning features. However the dimension of data set is 2048 which is too large to fit KNN model. To increase the efficiency of training, we use LDA to decrease the dimensions to 50. We divided the data set into training set and testing with 64 proportion. And use LDA model trained by training set to decrease the dimensions of both training set and testing set.

3.2 Simple Distance Metrics

3.2.1 Different P Values in Minkowski Distance

In Minkowski Distance, with different P values, different distance expressions will be formed. The knn function provided by sklearn can be adjusted, so we conducted experiments with different p values, and the results are as follows. From the results, it can be seen that the performance of the model first increases and then decreases with the increase of the P value, and the best performance results are obtained when p is between 1 and 2. The best performance we get is when $p = 1.5$ and $k = 26$, the accuracy is 0.9278.

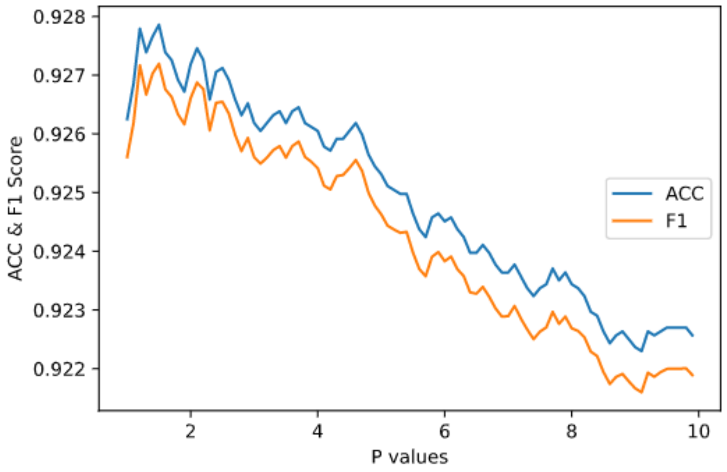


Figure 7: Performance with Different P Values

3.2.2 Different K Values and Simple Metrics

To find the best hyperparameters, we do experiments with different K-Values. The implementation of KNN uses interface provided by sklearn. The KNN algorithm provided by sklearn has built-in metric parameters, including chebyshev distance, euclidean distance, manhattan distance. However, cosine distance is not included. We can only implement it ourselves. And for the distance function is defined by ourselves with no optimization, the efficiency of fitting is low and it take almost 90 minutes for one testing. We test different k values and distance metric, the result is as follows

Table 1: Performance (ACC) of KNN with Different k Value and Metric

k	chebyshev	euclidean	manhattan	cosine
2	0.900931074	0.906892625	0.90481613	0.913323062
4	0.914595753	0.921361109	0.919016679	0.92316967
6	0.91740907	0.925447116	0.923370621	0.925916002
8	0.919351598	0.925179181	0.924911247	0.926987742
10	0.920155402	0.927121709	0.925581084	0.926987742
16	0.921227142	0.926853775	**0.926652823**	0.927121709
20	0.921160158	**0.927188693**	0.926250921	0.926786791
26	**0.921963963**	0.927188693	0.926183937	**0.927389644**
30	0.921896979	0.926719807	0.926317905	0.926786791
35	0.921428093	0.926719807	0.9255141	0.926451872
40	0.921227142	0.926183937	0.924308393	0.925782035
45	0.921026191	0.925849019	0.924107442	0.925849019
50	0.921026191	0.925380133	0.923839507	0.92658584

From the performance in the figure, both euclidean distance and cosine distance have good performance, chebyshev distance has the worst performance, and manhaten distance is between them. The

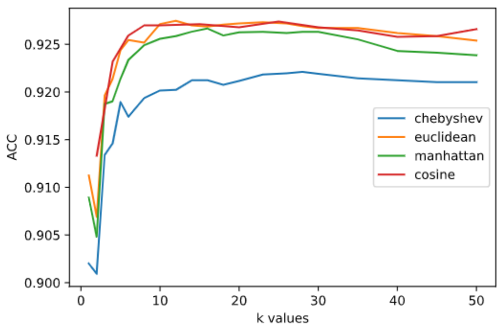


Figure 8: Different K and Distance (ACC)

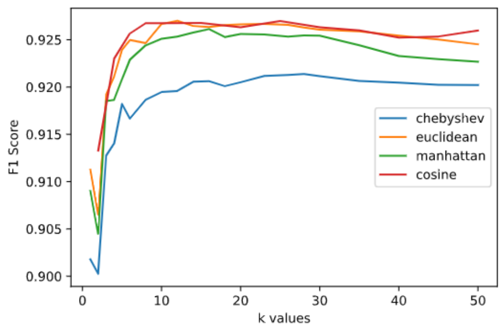


Figure 9: Different K and Distance (F1 Score)

classification accuracy rate is roughly the same with the k value: as the k value increases, there is a certain increase, but after it rises to a certain level, it gradually tends to be flat, and the increase is small. This is also roughly in line with the theoretical prediction. Since euclidean distance performs better and the fitting efficiency is high, subsequent experiments use euclidean distance and its optimal hyperparameters.

3.3 Metric Learning

3.3.1 MMC

The implementation of MMC still uses the interface provided by metric learn. The supervised version of MMCSupervised() is used here, where num constraints can be adjusted. It means number of constraints to generate.

Initially, using the default parameters performed poorly, with an accuracy rate of only about 0.89. Later, I tried to use a different number of constraints, and once again explored the number of K, and finally achieved better performance.

Table 2: Performance with Different Constrints of MMC

#constrints	ACC	F1 Score
200	0.919486	0.917981
400	0.920356	0.918414
600	0.922031	0.920148
700	**0.925447**	**0.924109**
800	0.92116	0.919761
900	0.92049	0.919075
1000	0.917677	0.91576
1500	0.913658	0.911989
2000	0.9225	0.921042
3000	0.920959	0.91892
4000	0.921093	0.919601
5000	0.912251	0.910947
1000	0.923438	0.921924
10000	0.922366	0.920335
100000	0.922433	0.920348
1000000	0.917744	0.915817

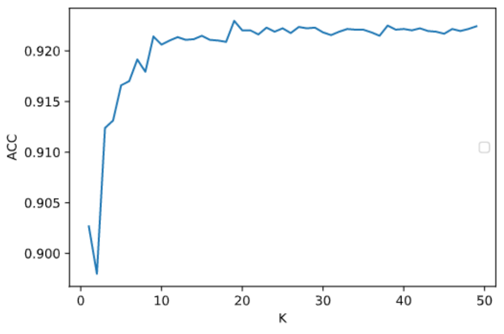


Figure 10: Different K

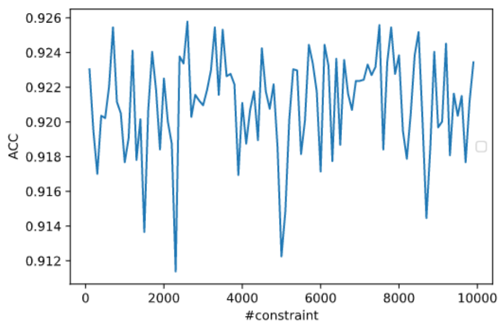


Figure 11: Different Constraints

From the experimental results, MMC did not provide a more obvious performance improvement, and the trend with k is roughly the same as that without metric learning. In addition, there is no obvious change in adjusting the constraint parameter within a larger data range, and there is just a fluctuation with the increase of the parameter.

We use the T-SNE method to visualize the data before and after the MMC, and we can see that the data discrimination is more obvious after the MMC. MMC does play a role in metric learning.

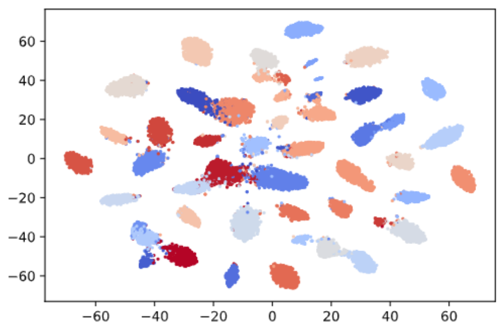


Figure 12: Before MMC (T-SNE)

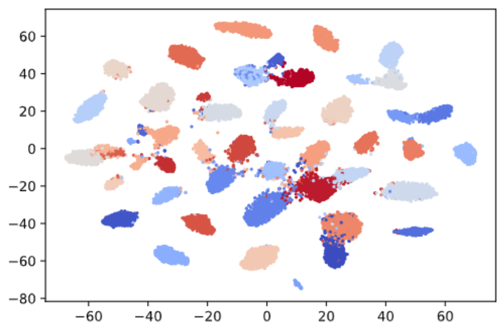


Figure 13: After MMC (T-SNE)

3.3.2 LMNN

Lmnn itself is based on K-nearest neighbor, k-nearest neighbor is completely consistent with KNN, that is to say, the two methods have the same idea, but the distance measurement methods are not the same, so when using lmnn, the same k-nearest neighbor can be set for lmnn and KNN. In this experiment, we choose not to set the same K in lmnn and KNN to observe what will happen.

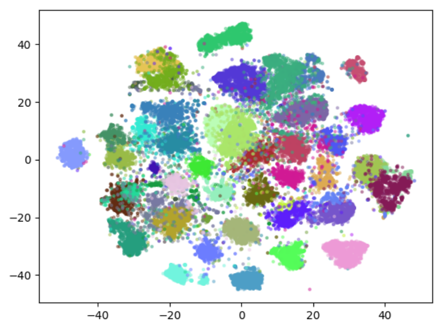


Figure 14: 32-d PCA

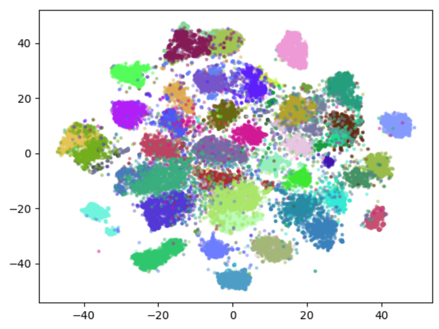


Figure 15: 128-d PCA

I choose to use k=1 to 12 in lmnn to get 12 lmnn models, and compare the performance of k = 1 to 32 in KNN. The k values of KNN and lmnn and the obtained ACC are plotted as follows

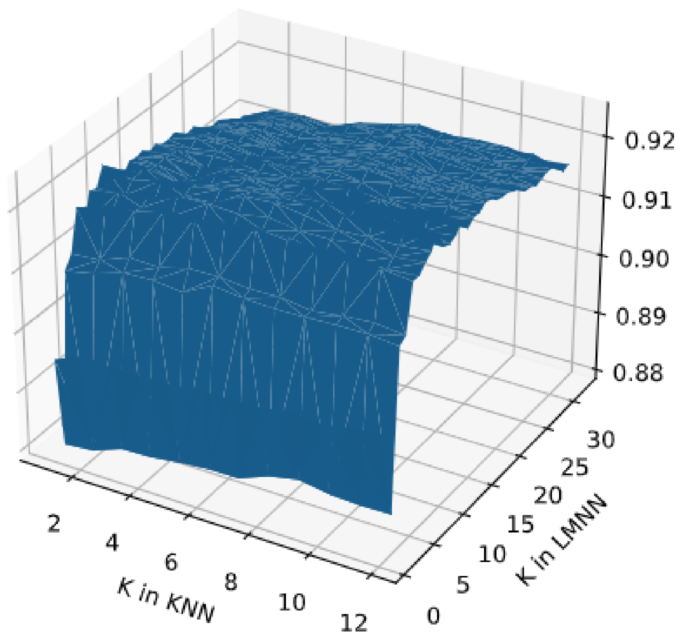
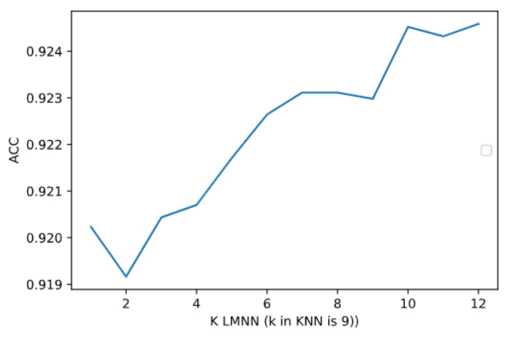
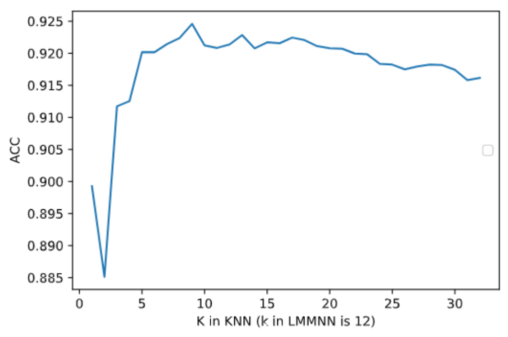


Figure 16: Performance with Different k Values in KNN and LMNN

Because when the K value of KNN is fixed, the change trend of ACC with the K value of lmnn is about the same. When the K value of lmnn is fixed, the change trend of ACC with the K value of KNN is about the same, so I choose some cases to make a two-dimensional image





At the same time, I also compare the working speed of lmnn in different dimensions. Here I choose the K value of KNN to take 16. I compare the time of lmnn with dimension 32 and dimension 128 under different K values

Table 3:

#time cost	32d	128d
k=3	442s	5201s
k=5	514	8320s
k=8	2120s	large

By observing and comparing the results, we can draw a conclusion: firstly, when k is even, compared with the odd K of adjacent selection, the performance will decrease significantly. So we suggest choosing an odd number as K. Second, when k is too large or too small, the performance will be degraded. We suggest choosing a suitable K for the experiment. Thirdly, when the k values of lmnn and KNN are equal or very close, the performance is the best, so we suggest to use the homogeneity of K values of lmnn and KNN to get better results. And as can be seen in the table, the time cost is increasing rapidly with the chosen k in LMNN and the dimension of samples. When k is larger than 8, we even couldnt get the result.

3.3.3 ITML

The implementation of ITML still uses the interface provided by metric learn. The supervised version of ITMLSupervised() is used here, where num constraints can be adjusted. It means number of constraints to generate.

Table 4: Performance with Different Constrints of ITML

#constrints	ACC	F1 Score
200	0.923571	0.923099
400	0.924174	0.923670
600	0.925128	0.924721
800	**0.925764**	**0.925109**
900	0.925294	0.925075
1000	0.924978	0.924387
4000	0.924777	0.924033
10000	0.923370	0.922582
100000	0.922499	0.921900

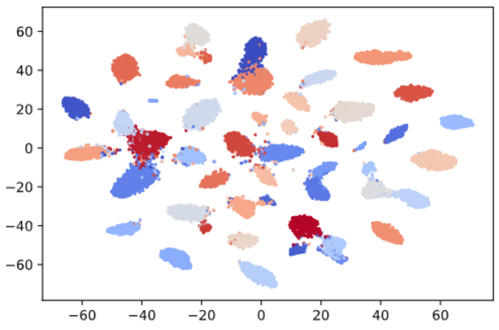


Figure 17: Before ITML (T-SNE)

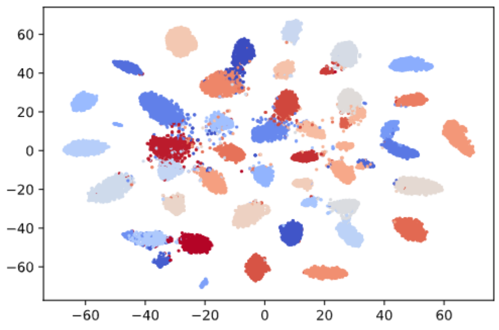


Figure 18: After ITML (T-SNE)

From the experimental results, itml does not provide more significant performance improvement, and the change of K value is almost the same as that of no metric learning. In addition, in a large range of data, the adjustment of constraint parameters has no obvious change.

At the same time, itml is the same as MMC, which is a supervised global metric learning method.

3.3.4 LSML

The implementation of LSML still uses the interface provided by metric learn. The supervised version of LSMLSupervised() is used here, where num constraints can be adjusted. It means number of constraints to generate.

From the experimental results, LSML does not provide more significant performance improvement, and the change of K value is almost the same as that of no metric learning. In addition, in a large range of data, the adjustment of constraint parameters has no obvious change.

Table 5: Performance with Different Constrints of LSML

#constrints	ACC	F1 Score
200	0.924169	0.923515
400	0.924392	0.923610
600	0.925181	0.924062
800	**0.925441**	**0.924576**
900	0.925382	0.925149
1000	0.925166	0.924521
4000	0.924948	0.924461
10000	0.923602	0.922930

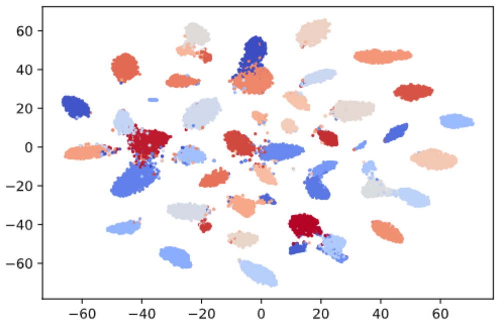


Figure 19: Before LSML (T-SNE)

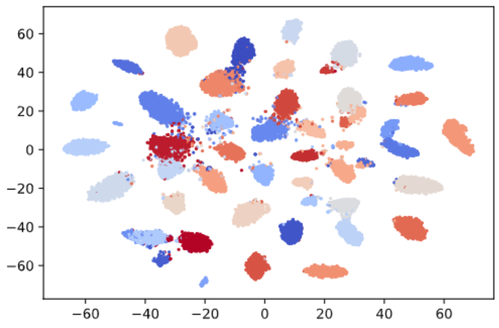


Figure 20: After LSML (T-SNE)

3.3.5 LDFA

I choose to use k=1 to 12 in LDFA to get 12 LDFA models, and compare the performance of k = 1 to 32 in KNN. The k values of KNN and LDFA and the obtained ACC are plotted as follows

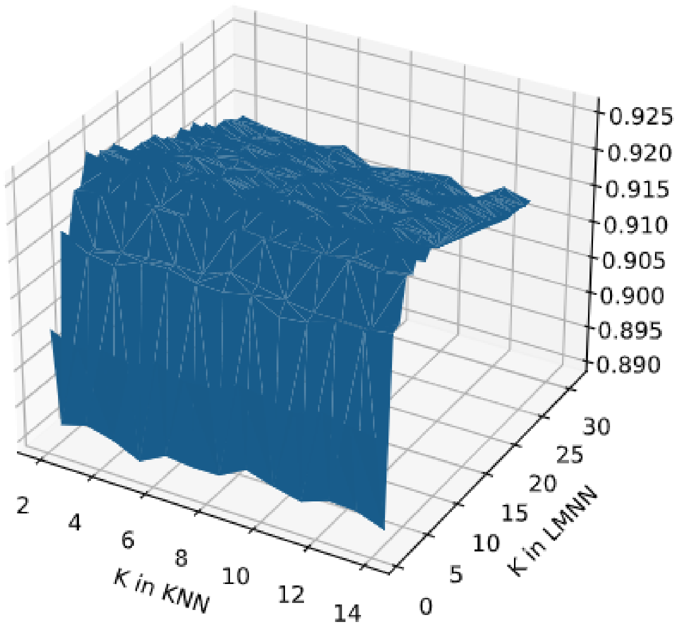
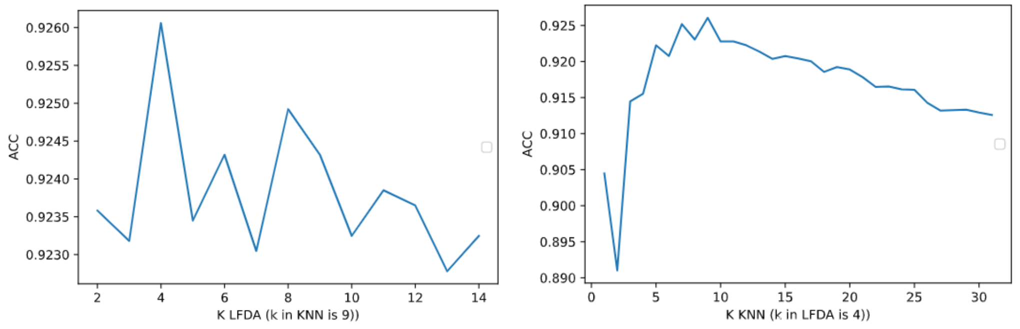


Figure 21: Performance with Different k Values in KNN and LDFA

Because when the K value of KNN is fixed, the change trend of ACC with the K value of LDFA is about the same. When the K value of LDFA is fixed, the change trend of ACC with the K value of KNN is about the same, so I choose some cases to make a two-dimensional image



Comparing these figures, we can see that when choosing different K in lfda, the performance trend of the model is not significantly different from that of lmnn, so we don't make too much summary here, but the performance of the model is worse than that of lmnn. We find that lfda is not only a metric learning method, but also a dimension reduction method, especially suitable for multimodal problems. So we can find that lfda may not be an appropriate method to solve this problem.

In addition, in the process of research, we find that lfda consumes less time, which is more time-saving. According to the analysis of lfda principle and related search, we conclude that the reasons for its less time consumption may be: first, ldfa algorithm may ignore some examples; second, the algorithm finds the local optimal solution, rather than the global optimal solution, It is possible to get a local optimal solution. This shows that lfda has a relatively free limit, but the effect may be slightly worse than lmnn.

3.3.6 RCA

In this section, firstly we compare different k in the RCA. We deployed PCA to reducing dimensionality to 32, and we get the following results:

Table 6: Performance with Different k of RCA(pre-processed by PCA)

#k	ACC	F1 Score
2	0.836527	0.832791
4	**0.870119**	**0.867049**
8	0.867074	0.865902
12	0.865124	0.864247

Then we deployed LDA to reducing dimensionality and compared with the following result. We found that the result by LDA is much better than that of PCA, shown as below:

Table 7: comparison between PCA and LDA

#k	LDA	PCA
4	**0.910045**	**0.870119**

It can be concluded that PCA lacks attention to categories when reducing dimensionality. So when RCA uses the information in such chunklets to reduce irrelevant variability in the data while amplifying relevant variability, it can not cooperate with PCA well.

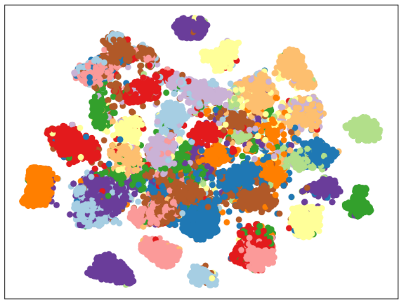


Figure 22: Before RCA (T-SNE)

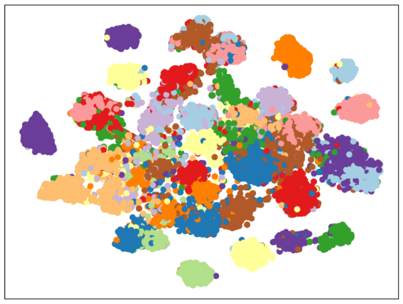


Figure 23: After RCA (T-SNE)

3.3.7 NCA

We deployed NCA by 32-component PCA dimensionality reductor. As we tested, the result varifying in K value in KNN are shown as below:

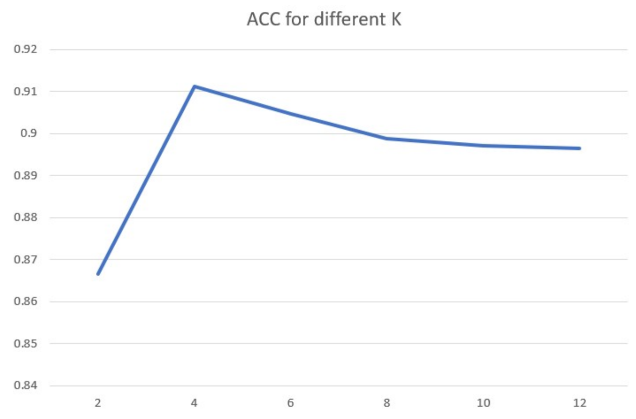


Figure 24: Performance with Different K Values in NCA

The method of NCA is not to consider K nearest neighbors in the new feature space, but to consider the entire data set as random nearest neighbors in the newspace. Therefore, when we consider k-nearest neighbors in the new feature space, the accuracy may be affected. So NCA is not a good choice for AwA2.

4 Conclusion

In this experiment, we study the principle and implementation of eight metric learning methods: lmnn, itml, sdml, lsml, NCA, lfda, RCA and MMC. And we adjust the parameters of different metric learning methods, and compare the performance of each different metric learning method to determine which K value is the best. We find that the optimal K value is basically the same for each method. At the same time, we find that some classical distance measures perform better than metric learning methods, and do not need to spend time on training and data. It also shows that the classical metric learning is still competitive in some data set features.

Also, by comparing several dimensional reduction methods, 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, RCA. Given the fact that when we train the model in RCA or LMNN and so on, training without any dimensional reduction will cause giant memory consumption, it is critical for us to use them. So we need to evaluate these features to gain the best performance.

Thanks

First of all, we would like to express our most sincere thanks and sincere admiration to our tutors and teaching assistants. Without the guidance and help of teachers and teaching assistants, we will not have the success of our thesis. Your meticulous and rigorous attitude provides us with a good academic research atmosphere. Secondly, we pay tribute to the authors and machine learning community workers who have helped us with the implementation of our paper. They have given us a lot of inspiration and help.

Table 8: Performance of Different Methods

Method	Parameter	ACC
Simple Metric (Minkowski)	p=1.5, k=26	0.9278
Simple Metric (Chebyshev)	k=26	0.9219
Simple Metric (Euclidean)	k=20	0.9271
Simple Metric (Manhattan)	k=16	0.9266
Simple Metric (Cosine)	k=26	0.9273
MMC	#constrints=700	0.9254
LMNN	k in LMNN=12	0.9245
ITML	#constrints=800	0.9257
LSML	#constrints=800	0.9254
LDFA	k=4	0.9260
NCA	k=4	0.9112
RCA	k=4 , using LDA	0.9100