CS245 Distance Metric

Yifan Lu, Yueyang Wu, Yuxin Lin

April 2021

Abstract

In this report, we do classification task on AwA2 dataset using K-Nearest-Neighbor algorithm. Based on the KNN method, we explored different metrics to measure the distance between samples, including various Minkowski distance, Cosine distance, etc. Also, we tried different metric learning method to do the task, including LMNN, LFDA, LSML, Covariance. Under each method, we do experiment on different parameters to attain the best performance.

1 Introduction

Distance metric is a key issue in many machine learning algorithms. Algorithms such as K Nearest Neighbor rely on the distance metric for the input data pattern heavily. Thus, how to measure the distance between samples and between distributions has received lots of attention. In this report we will talk about a number of techniques that is central to distance metric, including formula based distance and metric-learning based distance.

Metric learning 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. Many studies have shown that a learned metric can significantly improve the performance of many ML tasks, like clustering, classification, retrieval and so on. Depending on the availability of the training examples, algorithms for distance metric learning can be divided into two categories: supervised distance metric learning and unsupervised distance metric learning.

2 Dataset Preparation

AwA2[2] dataset consists of 37322 images of 50 animals classes. The author uses ResNet-101 to extract 2048-d deep learning feature to help training. However, we find it very time-comsuming when applying KNN on the total 2048 dim features, so we first reduce the feature dimensions as project 1 did.

The conclusion from project 1 tells us that LDA method reaches the best performance on feature reduction. So we first consider applying LDA on the original 2048 dim feature to 40

dimensions. We find it amazing that the naive KNN without any tricks can reach the accuracy of 0.93 on the test set, which might not be very good for us. We hope to see an increase in accuracy after using different metric learning methods, but the accuracy is high enough already, which will prevent us to observe the effect of metric learning.

We choose an alternative way to do dimensionality reduction. We apply PCA to reduce features to 40 dimensions. Now the highest accuracy of naive KNN on test set can be lower than 0.9, providing us an more acceptable baseline.

3 K Nearest Neighbor

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. Here KNN is the core of our experiment, which do the classification of 50 animal classes on AwA2 dataset.

The KNN algorithm assumes that similar things exist in close proximity. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

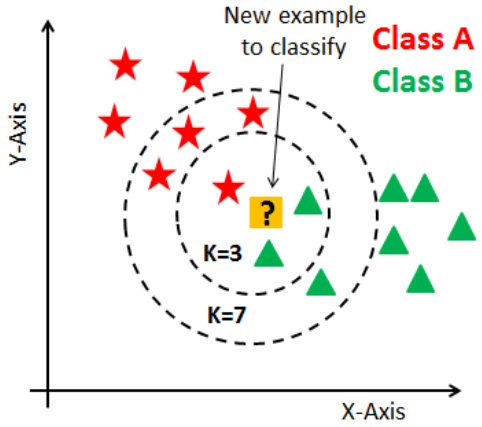

Figure 1: KNN Illustration

Take the picture above as an example, if we assign the number of neighbors (i.e. value for k) to be 3, then class B dominates and the data point will be assigned to be class B as well. If we choose k=7, then class A dominates and we put the data point to class A.

KNN is a lazy learning algorithm because it doesn't learn a discriminative function from the training data but "memorizes" the training dataset instead. There is no trainiing time in KNN, but the prediction time of KNN is relatively expensive. Each time we want to make a prediction,

KNN is searching for the nearest neighbors in the entire training set, so there are certain tricks such as BallTrees and KDtrees to speed this up.

We use `sklearn.neighbors.KNeighborsClassifier` in our experiment, we notice that we have a lot of parameters to choose, including:

1. **n_neighbors**(default=5) Number of neighbors to use by default for k neighbors queries.
2. **weights** (default='uniform') weight function used in prediction. Possible values:{'uniform', 'distance'}
3. **algorithm**(default='auto') Algorithm used to compute the nearest neighbors. Possible values:{'auto', 'ball_tree', 'kd_tree', 'brute'}. 'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method.
4. **p** (default=2) Power parameter for the Minkowski metric. When p = 1, this is equivalent to using Manhattan_distance (l_1), and Euclidean_distance (l_2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.
5. **metric** (default=Minkowski) The distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric.

4 Determine Hyperparameter

In the training set, we apply **k-fold cross validation** to determine the best parameters mentioned above. In this section, we will do experiment on **n_neighbors (or k)**, **weights** and **algorithm**, leaving the more detailed discussion about **p** and **metric** for subsequent parts.

4.1 k

k is the number of neighbors to vote. We find the chart has a obvious local minimum at k=2, and it is easy to explain: If the two neighbors belongs to different classes, then it becomes difficult for the algorithm to determine which class it really belongs to. So the accuracy of k=2 is lower than both k=1 and k=3, falling to a local minimum.

We find k=9 gains the best performance on validation set, and the optimal range for k should be from 3 to 15.

k	1	2	3	4	5	6	7	8	9	10
Acc	0.86931	**0.85974**	0.88348	0.88567	0.89027	0.89031	0.89143	0.89067	**0.89344**	0.89170
k	11	12	13	14	15	16	17	18	19	20
Acc	0.89250	0.89210	0.89107	0.89058	0.89179	0.89081	0.89027	0.88960	0.88946	0.88857

Table 1: K-Fold on find best k

4.2 weights

"weights" selects the weight function used in prediction. *'uniform'* means all points in each neighborhood are weighted equally. *'distance'* means weighting points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away. We try both weight function on different k, the result is shown below.

When choosing **weights**="distance", we can find the **accuracy at k=1 and k=2 are exactly the same**, because when two neighbors belongs to different class, and the closer one is assigned with higher weight, then it is equivalent to k=1 case.

k	1	2	3	4	5	6	7	8	9	10
Acc	0.86931	0.86931	0.88450	0.89121	0.89335	0.89487	0.89474	**0.89604**	0.89559	0.89545
k	11	12	13	14	15	16	17	18	19	20
Acc	0.89487	0.89496	0.89393	0.89483	0.89282	0.89335	0.89174	0.89157	0.89022	0.89045

Table 2: weights = "Distance", K-Fold validation

As shown in the chart, **weights**="distance" is a better strategy, showing better performance than the "uniform" choice. The optimal k under "distance" strategy is k=8, slightly different from the "uniform" case.

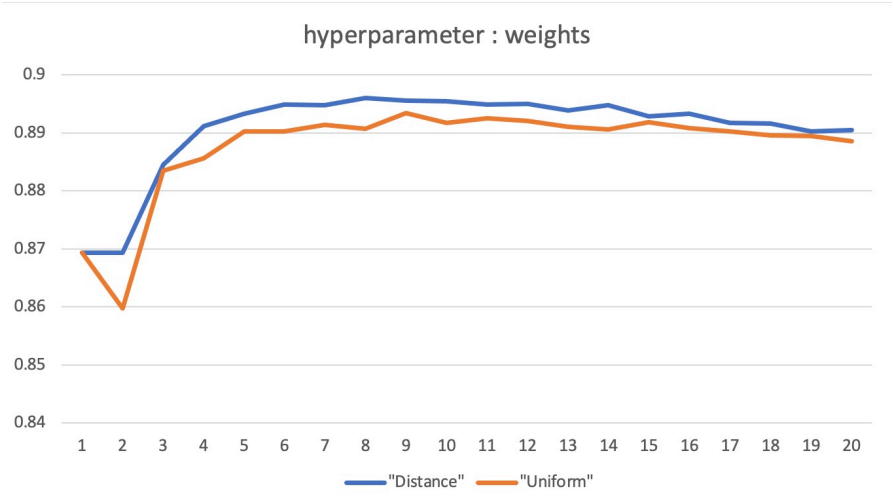

Figure 2: weights = "Distance" / "Uniform"

4.3 algorithm

we also explore the **algorithm** parameter in KNN, including {'auto', 'ball_tree', 'kd_tree', 'brute'}. Intuitively, different algorithms will lead to different running time without affecting the accuracy. Cause the distance is affected by the metrics.

algorithm	auto	Ball_tree	KD_tree	brute
running time(s)	319.89	346.00	318.66	**131.93**

Table 3: running time of K-fold validition, k from 1 to 20

We find the brute force gets the best running time performance, which is nearly 1/3 of other three algorithm. We think it is because the number of samples is relatively small, so the cost of maintaining data structures like Ball tree and KD tree is relatively expensive. As we expected, their accuracies are almost the same, so we can just use brute algorithm safely. Additionally, we find the strategy of **"auto" is exactly "KD Tree"**, in case of our feature data.

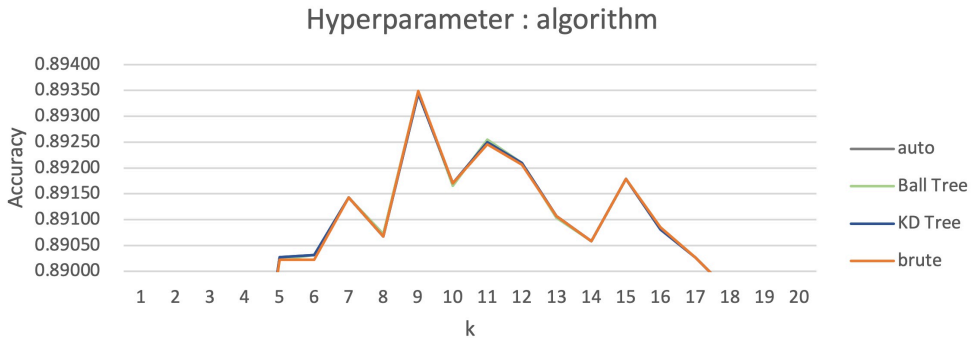

Figure 3: The accuracy has no difference

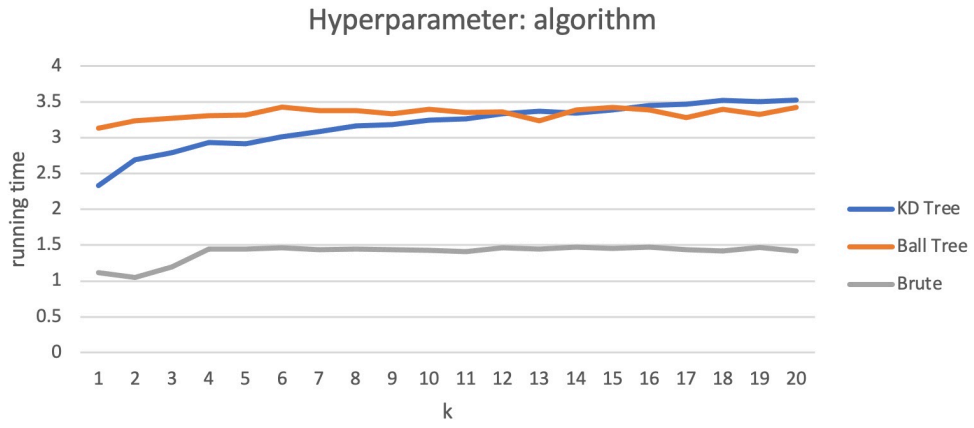

Figure 4: The running time differs a lot (one fold)

It's interesting to find that "brute" and "Ball Tree" algorithm's running time doesn't have much to do with k, except very small k values like k=1 or k=2. In contrast, the running time of "KD Tree" is positively correlated with K, shown as the blue line.

4.4 conclusion

To sum up, after our detailed studies on hyperparameters, we suggest using KNN with **weights**="distance", **algorithm**="brute", and **k** in range 3 to 15. In the subsequent experiment, we will also take this setting to be our baseline, comparing with different metric methods.

5 Different Simple Metric

5.1 Euclidean Distance

Euclidean distance reflects the linear distance between two points in Euclidean space. The formula for calculating Euclidean distance is:

$$d(x,y) = \sqrt{\sum_i (x_i - y_i)^2} \quad \text{or} \quad d(\vec{X} - \vec{Y}) = \sqrt{(\vec{X} - \vec{Y})^{\mathrm{T}}(\vec{X} - \vec{Y})}$$

It can also be regarded as L^2 norm of two vector difference.

5.2 Manhattan Distance

It comes from the distance among the streets of Manhattan. In Manhattan, where the block is very square, you can't get there directly from the starting point to the destination, but you have to go around the block at right angles. The formula for calculating Manhattan distance is:

$$d(x,y) = \sum_i |x_i - y_i|$$

Imagine taking a taxi from P to Q in Manhattan, where white represents tall buildings and gray represents streets:

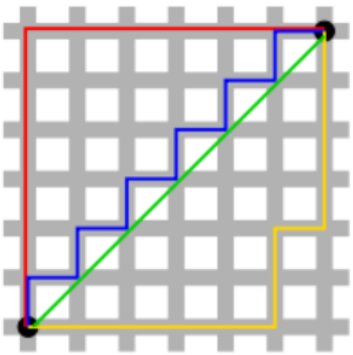

Figure 5: Manhattan Distance

The green slash indicates Euclidean distance, which is impossible in reality.The other three folds represent the distance from Manhattan, and the length of the three folds is the same.

5.3 Chebyshev Distance

The Chebyshev distance between two points is defined as the maximum value of the absolute value of the coordinate value difference. From the mathematical point of view, Chebyshev distance is a measure derived from uniform norm, and also a kind of hyperconvex measure. The formula for calculating Chebyshev Distance is:

$$d(x, y) = \max_i |x_i - y_i|$$

The Chebyshev distance between two positions on the chess board refers to the steps that Wang needs to take to move from one position to another. In chess, the king can go straight, sideways, or diagonally, so he can move to any of the eight adjacent squares with a single move.The minimum number of steps required for the king to go from the grid $(x_1, y1)$ to the grid (x_2, y_2) is always $Max(|x_2 - x_1|, |y_2 - y_1|)$ step. A similar measure of distance is called the Chebyshev distance (L_∞ norm).

Figure 6: Chebyshev distance

5.4 Minkowski Distance

Now let's talk about a more general case. The Minkowski distance is defined as follows:

$$d(x, y) = \left(\sum_i |x_i - y_i|^p \right)^{1/p} = \left(\sum_i |d_i|^p \right)^{1/p}$$

This is a broader norm form. It can be seen that Manhattan distance, Euclidean distance and Chebyshev distance can be regarded as the special form of Minkowski distance. When $p = 1$, this is equivalent to using manhattan distance (l_1), euclidean distance (l_2) for $p = 2$ and chebyshev distance when $p \to \infty$. For arbitrary p, minkowski distance (l_p) is used.

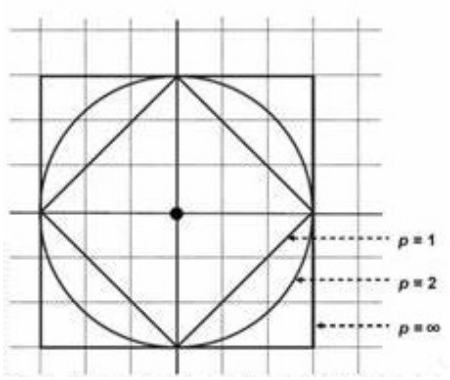

Figure 7: the value of p and distance

Figure 8: some other value of p and distance

Minkowski distance is more intuitive, but it has nothing to do with the distribution of data and has certain limitations. If the amplitude of the X direction is much greater than the value of the Y direction, this distance formula will over-amplify the effect of the X dimension. In short, the Minkowski distance has two main disadvantages:

1. The scale (unit) of each component is treated as the same.
2. It fails to consider that the distribution (expectation, variance, etc.) of each component may be different.

5.5 Cosine Distance

We can use cosine formula to calculate the similarity of two vectors. In a broad sense, similarity can be used to represent distance. But strictly speaking, it is not a standard distance measure. The formula for calculating cosine distance is:

$$\mathrm{Sim}_{\cos}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^{\mathrm{T}}\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$$

And when $\|x\| = 1, \|y\| = 1$ it has the following relationship with Euclidean distance:

$$d_{\text{euc}}\left(\mathbf{x}, \mathbf{y}\right) = \sqrt{2 - 2\,\text{sim}_{\cos}(\mathbf{x}, \mathbf{y})}$$

5.6 Mahalanobis Distance

Although the Euclidean distance we are familiar with is very useful, it has obvious disadvantages. It equates the differences between different properties of samples, which sometimes can not meet the actual requirements.

Mahalanobis distance is proposed to represent the distance between a point and a distribution. It is an effective method to calculate the similarity of two unknown sample sets. Different from Euclidean distance, it takes into account the relationship between various characteristics and is scale independent, that is, independent of the measurement scale. For a multivariable vector $x = (x_1, x_2, \cdots, x_p)^{\mathrm{T}}$ whose mean value is $\mu = (\mu_1, \mu_2, \cdots, \mu_p)^{\mathrm{T}}$ and covariance matrix is Σ, its Mahalanobis distance is:

$$D_M(x) = \sqrt{(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)}$$

Mahalanobis distance can also be defined as the degree of difference between two random variables X and Y that obey the same distribution and whose covariance matrix is Σ :

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x}-\vec{y})^{\mathrm{T}}\Sigma^{-1}(\vec{x}-\vec{y})}$$

As mentioned above, Minkowski distance is relatively intuitive, but it has nothing to do with the distribution of data and has certain limitations. If the dimensions are related to each other, this is where Mahalanobis distance comes in.This method uses the characteristics of data distribution to calculate different distances on the assumption that the dimensions of the data are not correlated. For example, consider the following diagram, where the ellipse represents the contour line, from Euclidean distance, the green-black distance is greater than the red-black distance, but from Mahalanobis distance, the result is the opposite:

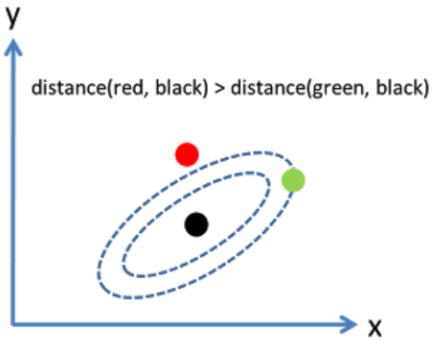

Figure 9: Mahalanobis distance

6 Experiment on Different Simple Distance Metrics

I use following distant metrics to carry on this experiment: Euclidean Distance Metrics, Manhattan Distance Metrics, Chebyshev Distance Metrics, Minkowski Distance Metrics(with p=3), Mahalanobis Distance Metrics, Cosine Distance Metrics. We set the KNN with **weights**="distance", **algorithm**="brute", as we explored in section 4.

Metric	Euclidean	Manhattan	Chebyshev	Minkowski	Mahalanobis	Cosine
Best_k	10	9	10	9	8	10
Acc.	**0.8880**	0.8853	0.8658	0.8855	0.8762	0.8794

Table 4: Different Simple Distance Metric

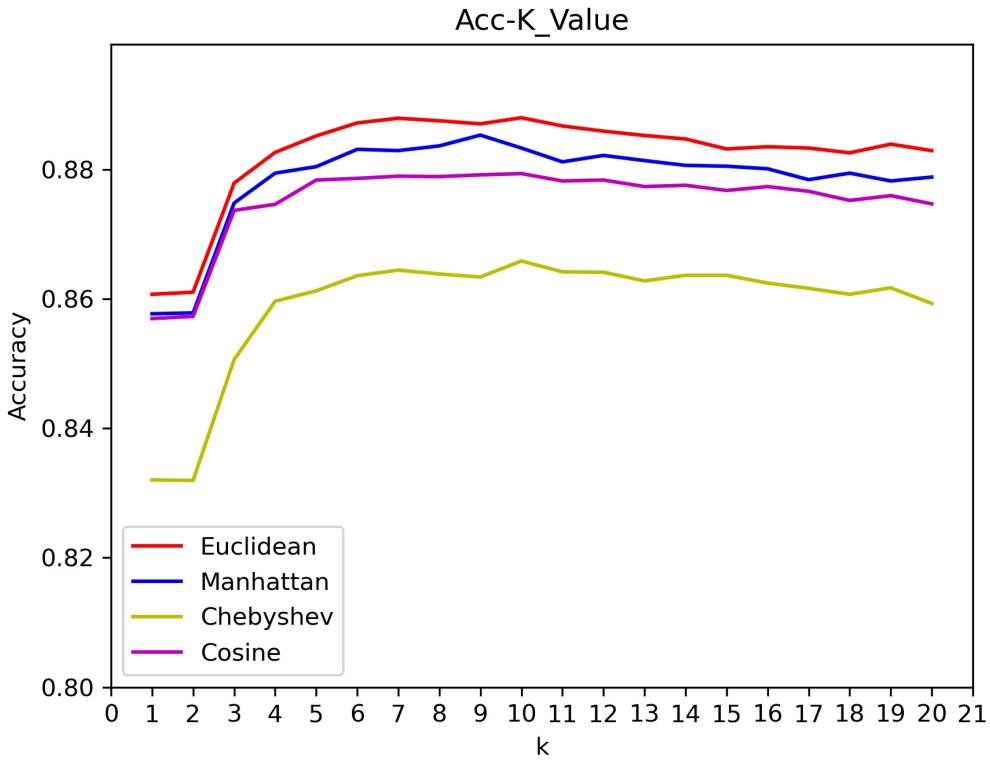

Figure 10: Different Simple Distance Metric

As showen in the table and figure, Euclidean Distance Metric performs better than Manhattan Distance Metric, Manhattan Distance Metric perform better than cosine Distance Metric. And Chebyshev Distance Metric has the worst performance.

7 Experiment on Different P-values in Minkowski Distance

Next, we do experiments to explore how the change of p-values in Minkowski distance would effect the results. In the former introduction of Minkowski distance, we can see that Minkowski distance can be regarded as the norm of vector difference.The larger the p value is, the fuller the envelope of norm in vector space is. We do experiment on p from 1 to 10.

p	1	2	3	4	5	6	7	8	9	10
Acc.(on Best_k)	0.8853	**0.8880**	0.8869	0.8821	0.8747	0.8634	0.8558	0.8504	0.8468	0.8434

Table 5: Different k in Minkowski Distance

We can tell from the result that, Minkowski Distance Metric showe the best performance when $p = 2$, and performance becomes worse as p get larger after $p = 2$. As p gets bigger, the fuller the envelope of norm in vector space is.

This is consistent with what we found above: Euclide Distance Metric performs better than Manhattan Distance Metric on this dataset, while Manhattan Distance Metric performs better than Chebyshev Distance Metric on this dataset.

8 Distance Metric Learning[1]

Although many standard distance metrics exist, It is still difficult to design metrics that are well-suited to the particular data and task of interest. To solve the problem, here we come up a set of methods called distance metric learning.

Distance metric learning (or simply, metric learning) aims at automatically constructing task-specific distance metrics from supervised data, in a machine learning manner. The learned distance metric can then be used to perform various tasks.

8.1 Supervised Metric Learning

Supervised metric learning algorithms (such as LMNN and LFDA) take as inputs points X and target labels y, and learn a distance matrix that make points from the same class (for classification) or with close target value (for regression) close to each other, and points from different classes or with distant target values far away from each other.

A variant is weakly supervised algorithm (such as LSML), which works on weaker information about the data points than supervised algorithms. Rather than labeled points, they take as input similarity judgments on tuples of data points, for instance pairs of similar and dissimilar points.

8.1.1 LMNN

LMNN learns a Mahalanobis distance metric in the kNN 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:

$$\min_{\mathbf{L}} \sum_{i,j} \eta_{ij} \left\| \mathbf{L}\left(\mathbf{x_i} - \mathbf{x_j}\right) \right\|^2 + c \sum_{i,j,l} \eta_{ij} \left(1 - y_{ij}\right) \left[1 + \left\| \mathbf{L}\left(\mathbf{x_i} - \mathbf{x_j}\right) \right\|^2 - \left\| \mathbf{L}\left(\mathbf{x_i} - \mathbf{x}_1\right) \right\|^2 \right]_+$$

where $\mathbf{x}_i$ is a data point, $\mathbf{x}_j$ is one of its k -nearest neighbors sharing the same label, and $\mathbf{x}_l$ are all the other instances within that region with different labels, $\eta_{ij}, y_{ij} \in \{0, 1\}$ are both the indicators, 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 classes, $[\cdot]_+ = \max(0, \cdot)$ is the Hinge loss.

8.1.2 LFDA

LFDA is a linear supervised dimensionality reduction method which effectively combines the ideas of Linear Discriminant Analysis and Locality-Preserving Projection. 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.

The algorithm define the Fisher local within-/between-class scatter matrix $\mathbf{S}^{(w)}/\mathbf{S}^{(b)}$ in a pairwise fashion:

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

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

where

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

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

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, n and n_l are the total number of points and the number of points per cluster l respectively.

Then the learning problem becomes derive the LFDA transformation matrix $\mathbf{L}_{LFDA}$:

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

That is, it is looking for a transformation matrix $\mathbf{L}$ 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.

8.1.3 LSML

LSML proposes a simple, yet effective, algorithm that minimizes a convex objective function corresponding to the sum of squared residuals of constraints. This algorithm uses the constraints in the form of the relative distance comparisons, such method is especially useful where pairwise constraints are not natural to obtain, thus pairwise constraints based algorithms become infeasible to be deployed. Furthermore, its sparsity extension leads to more stable estimation when the dimension is high and only a small amount of constraints is given.

The loss function of each constraint $d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{x}_k, \mathbf{x}_l)$ is denoted as:

$$H\left(d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) - d_{\mathbf{M}}(\mathbf{x}_k, \mathbf{x}_l)\right)$$

where $H(\cdot)$ is the squared Hinge loss function defined as:

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

The summed loss function $L(C)$ is the simple sum over all constraints:

$$C = \{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k, \mathbf{x}_l) : d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{x}_k, \mathbf{x}_l)\}$$

The distance metric learning problem becomes minimizing the summed loss function of all constraints plus a regularization term w.r.t. the prior knowledge:

$$\min_M D_{ld}(\mathbf{M}, \mathbf{M_0}) + \sum_{(\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d) \in C} H\left(d_{\mathbf{M}}(\mathbf{x}_a, \mathbf{x}_b) - d_{\mathbf{M}}(\mathbf{x}_c, \mathbf{x}_d)\right)$$

where $\mathbf{M}_0$ is the prior metric matrix, set as identity by default, $D_{ld}(\cdot, \cdot)$ is the LogDet divergence:

$$D_{ld}(\mathbf{M}, \mathbf{M_0}) = \operatorname{tr}(\mathbf{M}\mathbf{M_0}) - \log\det(\mathbf{M})$$

8.2 Unsupervised Metric Learning

Unsupervised metric learning algorithms only take as input an (unlabeled) dataset X. For now, in metric-learn, there only is Covariance, which is a simple baseline algorithm. For two jointly distributed real-valued random variables X and Y with finite second moments, the covariance is defined as the expected value (or mean) of the product of their deviations from their individual expected values:

$$\operatorname{cov}(X, Y) = \mathrm{E}[(X - \mathrm{E}[X])(Y - \mathrm{E}[Y])]$$

Covariance does not "learn" anything, rather it calculates the covariance matrix of the input data. This is a simple baseline method.

9 Experiment on Different Kinds of Metric Learning

- Covariance (Baseline): In order to compare the effectiveness of different kinds of metric learning, we need to set a baseline where we only use the simplest way, which is Covariance, to measure the distances. Here we use KNN classifier with data that has been reduced to 40 dimensions by PCA to avoid possible problems that may be caused by the heavy load. The parameters of the classifier are $weights = "distance", algorithm = "brute"$.
- LMNN: This method learns a Mahalanobis distance metric in the situation of KNN classification. The parameters we choose are $k = 3, min\_iter = 50, max\_iter = 1000, learn\_rate = 1e - 07$.
- LFDA: This is a linear supervised dimensionality reduction method. The parameters we choose are initial.
- LSML: Instead of original weakly supervised algorithm, we use a supervised version of LSML. The modified algorithm builds quadruplets by taking two samples from the same class while another two samples from different classes, and in that case, the two first points should be more similar than the two last points. Here we use $max\_iter = 1000$.

9.1 Comparison of the Effect

Detailed data and concrete line chart are shown in Fig.11 and Tab.6.

The purpose of metric learning is to find an appropriate way to calculate the similarity between different samples. We expect that the accuracy of the classification will be improved when the appropriate measure is found and be in use. Fortunately, as we can see, LMNN, LFDA, and LSML all performed better than the baseline. Among all of them, LMNN and LSML have about the similar overall effect, while LFDA performs slightly worse, which is possibly due to the conflict with PCA. As what we have mentioned, although being used in different situations and for different aims, LFDA and PCA are both dimensionality reduction methods.

In general, the classification performance of metric learning increases with the increase of k, and tends to be flat after $k \geq 7$ (also, a slight decline may exist). The k of the peak is not certain, but the accuracy does not fluctuate much when k is large. The best score for Covariance, LMNN, LFDA and LSML are 0.87611, 0.88942, 0.88508 and 0.89110, respectively. Even though the margin is very small, we can say that LSML performs the best.

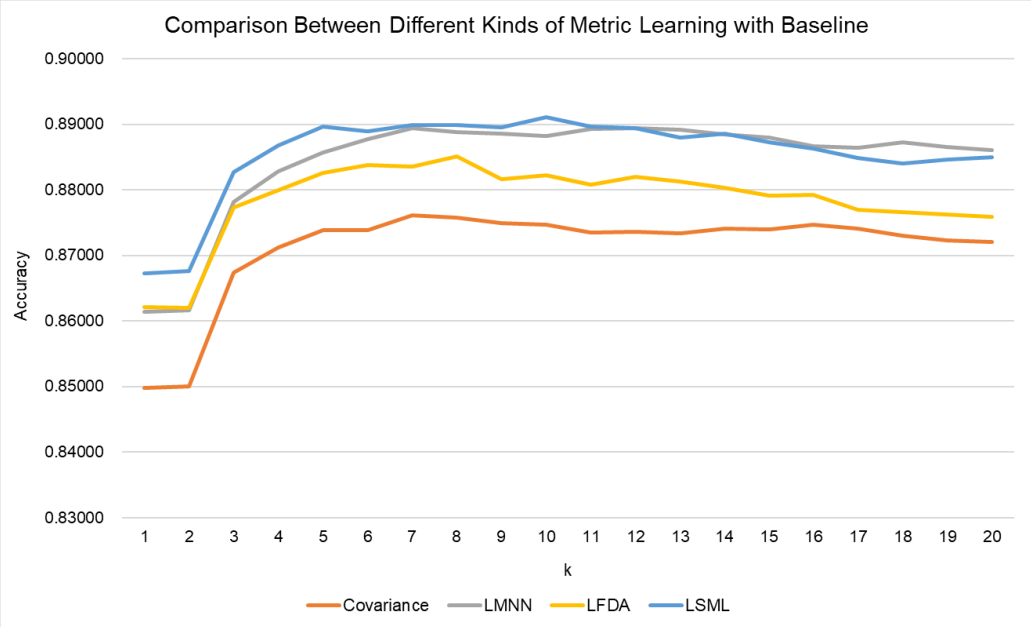

Figure 11: Comparison Between Different Kinds of Metric Learning

Table 6: Accuracy with different k

k	Covariance	LMNN	LFDA	LSML
1	0.84976	0.86140	0.86213	0.86722
2	0.85002	0.86160	0.86200	0.86762
3	0.86735	0.87819	0.87732	0.88267
4	0.87123	0.88287	0.87993	0.88675
5	0.87384	0.88574	0.88260	0.88962
6	0.87390	0.88768	0.88374	0.88896
7	**0.87611**	0.88936	0.88354	0.88989
8	0.87571	0.88876	**0.88508**	0.88989
9	0.87491	0.88855	0.88160	0.88956
10	0.87464	0.88822	0.88227	**0.89110**
11	0.87350	0.88929	0.88079	0.88962
12	0.87357	**0.88942**	0.88193	0.88942
13	0.87337	0.88922	0.88126	0.88802
14	0.87411	0.88849	0.88033	0.88855
15	0.87397	0.88795	0.87912	0.88728
16	0.87471	0.88668	0.87926	0.88628
17	0.87411	0.88641	0.87691	0.88481
18	0.87297	0.88722	0.87658	0.88407
19	0.87230	0.88655	0.87618	0.88461
20	0.87210	0.88608	0.87591	0.88501

9.2 Comparison of the Time Consuming

In fact, the time consumption of different kinds of metric learning is clear at a glance, as what we have shown in Tab.7.

The baseline consumed the least time, while the LMNN consumed 1833.77s, which is more than 10 times longer than the baseline. Excluding the time spent on classification and other work and only taking the time spent on metric learning into account, The LFDA and LSML are fast enough that they actually consume only about 3s and 6s more than the baseline. Considering the accuracy we have given in the previous section, we can say that LSML is the best overall.

Table 7: Time Consuming

Method	Covariance	LMNN	LFDA	LSML
Time(s)	129.67867	1833.76750	132.62804	135.93403

10 Conclusion

In this project, we explore different distance metric on reduced AwA2 dataset. We also find the best hyperparameter setting of KNN to do the classification task. We explain the principle of Euclidean distance, Manhattan distance, Chebyshev distance, Minkowski distance, Cosine distance, Covariance, LMNN, LFDA, LSML in detail, and do experiment on all of them. Among simple distance metric methods, Euclidean distance achieves the highest accuracy 0.8880. Metric Learning method can indeed improve the classification performance, in our experiment, LSML get higher accuracy 0.8911.

References

[1] William de Vazelhes Aurélien Bellet CJ Carey, Yuan Tang and Nathalie Vauquier. metric-learn: Metric learning in python.

[2] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2251–2265, 2019.