KNN Classification with Different Distance Metrics

Yanjun Fu 516030910354
Zeren Huang 516030910356
Hao Sun 516030910362
Bohong Wu 516030910365

1. Introduction

In this project, we need to categorize a large number of images into 50 different classes. To avoid the heavy computational overhead of extracting features from images, all of the images were pre-extracted into 2048-dimensional features before.

This project is mainly divided into two parts. First, We use K-Nearest Neighbors(KNN) classification with 8 distance metrics [1] such as Euclidean distance, Chebyshev distance and Canberra distance, and we compare their performance to explore which metric can achieve the optimal performance. Second, we use several metric learning methods such as LMNN, NCA, and MLKR [5] to learn a good metric, which can help improve the performance of KNN.

2. Methods

In this section, we will introduce the definitions of different distance metrics and the basic theories of metric learning.

2.1. K-nearest Neighbors Classification

Neighbors-based classification is a type of instance-based learning or non-generalizing learning, it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

The basic nearest neighbors classification uses uniform weights, that is, the value assigned to a query point is computed from a simple majority vote of the nearest neighbors. Under some circumstances, it is better to weight the neighbors such that nearer neighbors contribute more to the fit. Here we call the two methods as **"hard-KNN"** and **"soft-KNN"**. In the experiment, we adopted two methods and then compared their performance.

2.2. Steps of KNN Algorithm

1. Store all training data which has been labeled.
2. For each sample $\mathbf{x}$ that need to be predicted, compute the distances between $\mathbf{x}$ and all other training data.
3. Sort the distance list in ascending order and get the k nearest data.
4. **Hard-KNN:** Sample $\mathbf{x}$ is predicted as the mode of the labels of those k nearest data.

   Soft-KNN: Sample $\mathbf{x}$ is predicted as the weighted mode of the labels of those k nearest data. The weight assignment is to weight 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.

An illustration of KNN is shown in fig. 1.

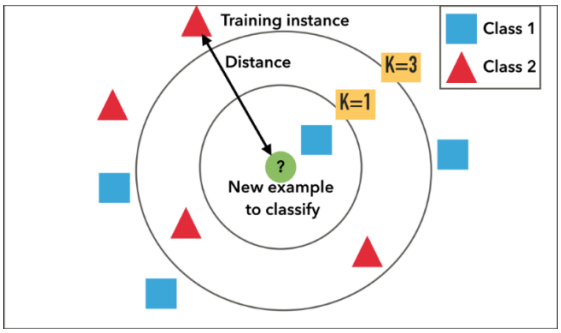

Figure 1. An Illustration for KNN

2.3. Distance Metrics

Given two vectors:

$$\mathbf{x} = [x_1, x_2, x_3, \cdots, x_n]^T, \mathbf{y} = [y_1, y_2, y_3, \cdots, y_n]^T,$$

the distance between $\mathbf{x}$ and $\mathbf{y}$, $d(\mathbf{x}, \mathbf{y})$, can be defined in many ways.

2.3.1 Minkowski Distance

$$d(\mathbf{x}, \mathbf{y}) = \left[\sum_i (x_i - y_i)^p\right]^{\frac{1}{p}} \tag{1}$$

When p is given a specific value, minkowski distance is equivalent to other distance metric as follows.

- When $p = \infty$, it is Chebyshev distance: $d(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|$
- When $p = 2$, it is Euclidean distance: $d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_i (x_i - y_i)^2}$
- When $p = 1$, it is Manhattan distance: $d(\mathbf{x}, \mathbf{y}) = \sum_i |x_i - y_i|$

2.3.2 Cosine Distance

$$d(\mathbf{x}, \mathbf{y}) = 1 - \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \tag{2}$$

2.3.3 Hamming Distance

$$d(\mathbf{x}, \mathbf{y}) = \frac{\sum_i^N \mathrm{NEQ}(x_i, y_i)}{N}, \tag{3}$$

where N represents the dimension of vector $\mathbf{x}$ and $\mathbf{y}$, and the definition of function NEQ is:

$$\mathrm{NEQ}(p, q) = \begin{cases} 1 & p \neq q \\ 0 & p = q \end{cases} \qquad p, q \in \mathbf{R}$$

2.3.4 Canberra Distance

$$d(\mathbf{x}, \mathbf{y}) = \sum_i \frac{|x_i - y_i|}{|x_i| + |y_i|} \tag{4}$$

2.3.5 Braycurtis Distance

$$d(\mathbf{x}, \mathbf{y}) = \frac{\sum_i |x_i - y_i|}{\sum_i |x_i| + \sum_i |y_i|} \tag{5}$$

2.3.6 Standard Euclidean Distance

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^N \frac{(x_i - y_i)^2}{s_i^2}}, \tag{6}$$

where s_i is the variance of the i^{th} dimension of all data.

2.3.7 Mahalanobis Distance

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mathbf{y})}, \tag{7}$$

where $\mathbf{\Sigma}$ is the covariance matrix of samples.

2.4. Metric Learn

Distance metrics are widely used in the machine learning literature. Traditionally, practitioners would choose a standard distance metric (Euclidean, City-Block, Cosine, etc.) using a priori knowledge of the domain. Distance metric learning (or simply, metric learning) is the sub-field of machine learning dedicated to automatically constructing optimal distance metrics.

Suppose we have some sets of points $\{\mathbf{x}_i\}_{i=1}^m \subseteq \mathbb{R}^n$, and are given information that certain pairs of them are similar.

$$S: \quad (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S} \quad \text{if } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are similar.} \tag{8}$$

Consider learning a distance metric of the form

$$d(\mathbf{x}, \mathbf{y}) = d_{\mathbf{A}}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{A} (\mathbf{x} - \mathbf{y})}. \tag{9}$$

A simple way of defining a criterion for the desired metric is to demand that pairs of points $(\mathbf{x_i}, \mathbf{x_j})$ in $\mathcal{S}$ have, say, small squared distance between them:

$$\mathrm{minimize}_{\mathbf{A}} \sum_{(\mathbf{x_i}, \mathbf{x_j}) \in \mathcal{S}} \|\mathbf{x_i} - \mathbf{x_j}\|_{\mathbf{A}}^2 .$$

This is trivially solved with $\mathbf{A} = \mathbf{0}$, which is not useful, and we add the constraint

$$\sum_{(\mathbf{x_i}, \mathbf{x_j}) \in \mathcal{D}} \|\mathbf{x_i} - \mathbf{x_j}\|_{\mathbf{A}} \geq 1$$

to ensure that $\mathbf{A}$ does not collapse the dataset into a single point. Here, $\mathcal{D}$ can be a set of pairs of points known to be dissimilar if such information is explicitly available; otherwise, we may take it to be all pairs not in $\mathcal{S}$. This gives the optimization problem:

$$\begin{aligned} \min_{\mathbf{A}} & \sum_{(\mathbf{x_i}, \mathbf{x_j}) \in \mathcal{S}} \|\mathbf{x_i} - \mathbf{x_j}\|_{\mathbf{A}}^2 \\ \text{s. t.} & \sum_{(\mathbf{x_i}, \mathbf{x_j}) \in \mathcal{D}} \|\mathbf{x_i} - \mathbf{x_j}\|_{\mathbf{A}} \geq 1 \\ & \mathbf{A} \succeq \mathbf{0} \end{aligned} \tag{10}$$

2.4.1 Large Margin Nearest Neighbor (LMNN)

LMNN [6] learns a Mahanalobis distance metric in the KNN classification setting using semi-definite programming. The learned metric attempts to keep k-nearest neighbors in 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.

Large margin nearest neighbors optimizes the matrix $\mathbf{M}$ with the help of semi-definite programming. For every data point $\mathbf{x}_i$, the target neighbors should be close and the impostors should be far away. The fig. 2 shows the effect

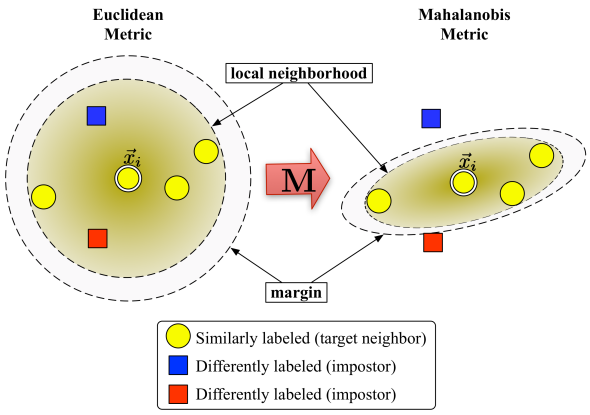

Figure 2. Schematic Illustration for LMNN

of such an optimization on an illustrative example. The learned metric causes the input vector $\mathbf{x}_i$ to be surrounded by training instances of the same class. If it was a test point, it would be classified correctly under the $k = 3$ nearest neighbor rule.

The first optimization goal is achieved by minimizing the average distance between instances and their target neighbors

$$\sum_{i,j \in N_i} d\left(\mathbf{x}_i, \mathbf{x}_j\right)$$

The second goal is achieved by penalizing distances to impostors $\mathbf{x}_i$ that are less than one unit further away than target neighbors $\mathbf{x}_i$ (and therefore pushing them out of the local neighborhood of $\mathbf{x}_i$). The resulting value to be minimized can be stated as:

$$\sum_{i,j \in N_i, l, y_l \neq y_i} \left[d\left(\mathbf{x}_i, \mathbf{x}_j\right) + 1 - d\left(\mathbf{x}_i, \mathbf{x}_l\right)\right]_+$$

With a hinge loss function $[\cdot]_+ = \max(\cdot, 0)$, which ensures that impostor proximity is not penalized when outside the margin. The margin of exactly one unit fixes the scale of the matrix $\mathbf{M}$. Any alternative choice $c > 0$ would result in a rescaling of $\mathbf{M}$ by a factor of $1/c$.

The final optimization problem becomes:

$$\begin{aligned} \min_{\mathbf{M}} & \sum_{i,j \in N_i} d\left(\mathbf{x}_i, \mathbf{x}_j\right) + \lambda \sum_{i,j,l} \xi_{ijl} \\ & \forall_{i,j \in N_i, l, y_i \neq y_i} \\ \text{s.t. } & d\left(\mathbf{x}_i, \mathbf{x}_j\right) + 1 - d\left(\mathbf{x}_i, \mathbf{x}_l\right) \leq \xi_{ijl} \\ & \xi_{ijl} \geq 0 \\ & \mathbf{M} \succeq 0 \end{aligned} \tag{11}$$

The hyperparameter $\lambda > 0$ is some positive constant (typically set through cross-validation). Here the variables ξ_{ijl} (together with two types of constraints) replace the term in the cost function. They play a role similar to slack variables to absorb the extent of violations of the impostor constraints. The last constraint ensures that $\mathbf{M}$ is positive semi-definite. The optimization problem is an instance of semi-definite programming (SDP). Although SDPs tend to suffer from high computational complexity, this particular SDP instance can be solved very efficiently due to the underlying geometric properties of the problem. In particular, most impostor constraints are naturally satisfied and do not need to be enforced during runtime (i.e. the set of variables ξ_{ijl} is sparse). A particularly well suited solver technique is the working set method, which keeps a small set of constraints that are actively enforced and monitors the remaining (likely satisfied) constraints only occasionally to ensure correctness.

2.4.2 Neighborhood Components Analysis (NCA)

Neighbourhood components analysis [3] is a supervised learning method for classifying multivariate data into distinct classes according to a given distance metric over the data. Functionally, it serves the same purposes as the K-nearest neighbors algorithm, and makes direct use of a related concept termed stochastic nearest neighbours.

we restrict ourselves to learning Mahalanobis (quadratic) distance metrics, which can always be represented by symmetric positive semi-definite matrices. We estimate such metrics through their inverse square roots, by learning a linear transformation of the input space such that in the transformed space, KNN performs well. If we denote the transformation by a matrix A we are effectively learning a metric $\mathbf{Q} = \mathbf{A}^T\mathbf{A}$ such that

$$d(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T \mathbf{Q} (\mathbf{x} - \mathbf{y}) = (\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{y})^T (\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{y})$$

.

The objective we maximize is the expected number of points correctly classified under this scheme:

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

where we define the p_{ij} using a softmax over Euclidean distances in the transformed space.

$$p_{ij} = \frac{\exp\left(-\left\|\mathbf{A}\mathbf{x}_i - \mathbf{A}\mathbf{x}_j\right\|^2\right)}{\sum_{k \neq i} \exp\left(-\left\|\mathbf{A}\mathbf{x}_i - \mathbf{A}\mathbf{x}_k\right\|^2\right)} \quad , \quad p_{ii} = 0 \tag{13}$$

Maximizing the objective function $f(A)$ is equivalent to minimizing the $L1$ norm between the true class distribution (having probability one on the true class) and the stochastic class distribution induced by p_{ij} via A. A natural alternative distance is the KL-divergence which induces the following

objective function:

$$g(A) = \sum_i \log\left(\sum_{j \in C_i} p_{ij}\right) = \sum_i \log\left(p_i\right) \tag{14}$$

2.4.3 Metric Learning for Kernel Regression (MLKR)

MLKR is an algorithm for supervised metric learning, which learns a distance function by directly minimising the leave-one-out regression error. This algorithm can also be viewed as a supervised variation of PCA and can be used for dimensionality reduction and high dimensional data visualization.

Our approach applies to any distanced-based kernel function $k\left(\mathbf{x}_i, \mathbf{x}_j\right) = k_D\left(D_\theta\left(\mathbf{x}_i, \mathbf{x}_j\right)\right)$ with differential dependence on parameters specifying the distance function D_θ. Specifically, MLKR consists of setting initial values of θ, and then adjusting the values using a gradient descent procedure:

$$\Delta\theta = -\epsilon \frac{\partial \mathcal{L}}{\partial \theta} \tag{15}$$

where ϵ is an adaptive step-size, and the loss function $\mathcal{L}$ is the cumulative leave-one-out quadratic regression error of the training samples:

$$\mathcal{L} = \sum_i \left(y_i - \hat{y}_i\right)^2 \tag{16}$$

with $\hat{y}_i$ defined as

$$\hat{y}_i = \frac{\sum_{j \neq i} y_j k_{ij}}{\sum_{j \neq i} k_{ij}} \tag{17}$$

where $k_{ij} = k\left(\mathbf{x}_i, \mathbf{x}_t\right) \geq 0$ is referred to as the kernel function. A wide variety of kernel functions have been studied in prior literature, e.g., Gaussian, triangular, spherical, wave, etc..

3. Experiments and Results

In this section, we will first simply present the experiment procedure and settings. To evaluate different traditional distance metrics and metric learning methods previously described, we will present our experiment results and give a comprehensive comparison and analysis.

3.1. Data Processing

1. Download AwA2 dataset from https://cvml.ist.ac.at/AwA2/. This dataset consists of 37322 images of 50 animal classes with pre-extracted deep learning features for each image.

2. Split the images of each category into 60% for training and 40% for testing.

3.2. Hyper-parameter Settings

In KNN algorithm, we could tune the hyper-parameter k to yield a better classification result. In our experiment, we change k from 3 to 20 and conduct experiments respectively. For each distance metric, we would show the results under different ks and finally decide the optimal value of k.

3.3. Baseline

To make a better comparison, we regard hard KNN under **Euclidean distance** as the baseline method. When hyper-parameter k is set as 9, we get the highest $\mathrm{accuracy} = 0.89705$.

3.4. Results of Different Distance Metrics

3.4.1 Euclidean Distance

Euclidean distance is usually used as the default distance metric in the KNN algorithm, so it is quite reasonable to regard it as the baseline. It is also a special case of Minkowski distance when $p = 2$.

We conduct experiments with different hyper-parameter ks and we will show both the classification results using hard-KNN and soft-KNN. The relative results are shown in table 1, and the curves of accuracy with different ks are shown in fig. 3.

Table 1. The Results of Euclidean distance

k	Accuracy of Hard-KNN	Accuracy of Soft-KNN
3	0.89028	0.88921
4	0.89055	0.89108
5	0.89611	0.89088
6	0.89423	0.89236
7	0.89698	0.89155
8	0.89564	**0.89303**
9	**0.89705**	**0.89303**
10	0.89631	0.89209
11	0.89584	0.89068
12	0.89417	0.89102
13	0.89376	0.88934
14	0.89316	0.88800
15	0.89256	0.88666
16	0.89276	0.88619
17	0.89162	0.88412
18	0.89062	0.88452
19	0.88961	0.88452
20	0.88861	0.88244

For hard-KNN, when k is set to 9, we get the highest $\mathrm{accuracy} = 0.89705$; for soft-KNN, when k is set to 8 or 9, we get the highest $\mathrm{accuracy} = 0.89236$.

In general, KNN using Euclidean distance performs well in this task and gets relatively high accuracy compared to

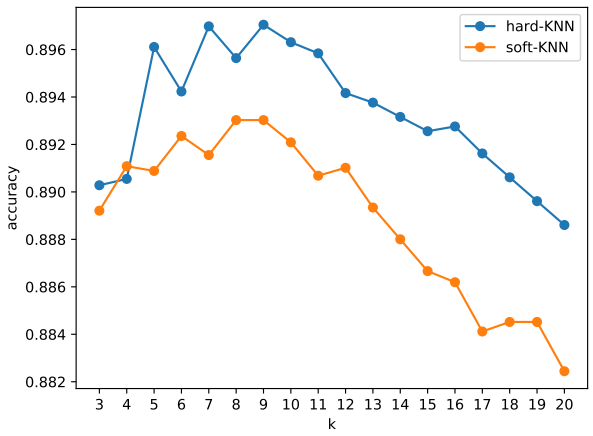

Figure 3. Results of Euclidean Distance with Different ks

most of the distance metrics we have introduced, and in many other cases its performance is also barely satisfactory.

However, under the situation that not all the dimensions of data have the same units, Euclidean distance would probably not make sense since it just simply adds all the squared value of them. In this experiment, all dimensions of pre-extracted deep learning features have the same unit and can be treated equally, so using Euclidean distance is quite reasonable.

3.4.2 Chebyshev Distance

Chebyshev distance is a special case of Minkowski distane when $p = \infty$. It is a metric defined on a vector space where the distance between two vectors is the greatest of their differences along any coordinate dimension. The relative resuls are shown in table 2, and the curves of accuracy with different ks are shown in fig. 4.

For hard-KNN, when k is set to 7, we get the highest $\text{accuracy} = 0.78284$; for soft-KNN, when k is set to 6, we get the highest $\text{accuracy} = 0.79476$.

Compared to the baseline metric, Chebyshev distance has poorer performance. For all of ks we have experimented, the accuracy of hard-KNN and soft-KNN are both less than 0.80. On the other words, Chebyshev distance does not work in the KNN algorithm compared to some of other metrics.

The reason why Chebyshev distance performs not so well in this case is probably because that it only considers the dimension which has the highest difference, thus neglects other dimensions which also contains much useful information which can help classification.

Table 2. The Results of Chebyshev distance

k	Accuracy of Hard-KNN	Accuracy of Soft-KNN
3	0.76710	0.78451
4	0.77734	0.78927
5	0.77842	0.78987
6	0.78096	**0.79476**
7	**0.78284**	0.79302
8	0.78197	0.79423
9	0.78244	0.79336
10	0.78210	0.79349
11	0.78103	0.79195
12	0.78257	0.79181
13	0.77922	0.78887
14	0.77855	0.78974
15	0.77681	0.78713
16	0.77587	0.78739
17	0.77507	0.78706
18	0.77426	0.78592
19	0.77386	0.78505
20	0.77312	0.78438

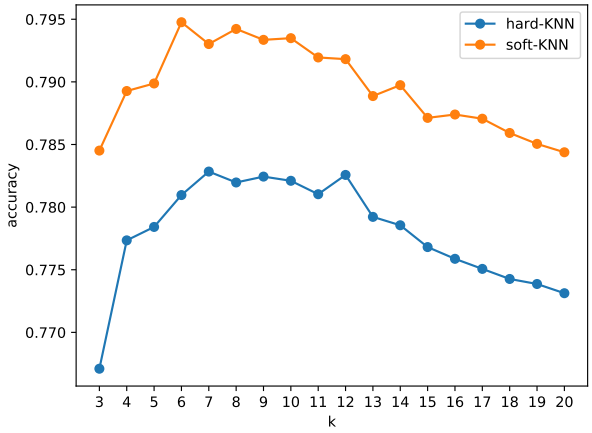

Figure 4. Results of Chebyshev distance with Different ks

3.4.3 Manhattan Distance

Manhattan distance is also a special case of Minkowski distance when $p = 1$. It measures the distance between two points with the sum of the absolute differences of their coordinates. It is especially useful in "Compressed Sensing" and "Chessboard Measuring". The relative results are shown in table 3, and the curves of accuracy with different ks are shown in fig. 5.

For hard-KNN, when k is set to 5, we get the highest $\text{accuracy} = 0.87930$, while for soft-KNN, when k is set to 2, we get the highest $\text{accuracy} = 0.89088$.

Similarly, Manhattan Distance is reasonable only when

Table 3. The Results of Manhattan Distance

k	Accuracy of Hard-KNN	Accuracy of Soft-KNN
3	0.87829	0.88867
4	0.87494	**0.89088**
5	**0.87930**	0.88934
6	0.87561	0.88673
7	0.87802	0.88512
8	0.87662	0.88552
9	0.87695	0.88385
10	0.87394	0.88459
11	0.87454	0.88331
12	0.87286	0.88077
13	0.87420	0.88050
14	0.87327	0.88117
15	0.87286	0.87903
16	0.87153	0.87822
17	0.87119	0.87601
18	0.87019	0.87508
19	0.86952	0.87407
20	0.86757	0.87387

Table 4. The Results of Standard Euclidean Distance

k	Accuracy of Hard-KNN	Accuracy of Soft-KNN
3	0.88559	0.88921
4	0.88512	**0.89336**
5	**0.88707**	0.89303
6	0.88653	0.89323
7	0.88700	0.89122
8	0.88398	0.89182
9	0.88372	0.89075
10	0.88385	0.89102
11	0.88318	0.88861
12	0.88110	0.88907
13	0.88177	0.88552
14	0.87942	0.88593
15	0.87936	0.88599
16	0.87882	0.88646
17	0.87782	0.88439
18	0.87648	0.88285
19	0.87628	0.88084
20	0.87474	0.88124

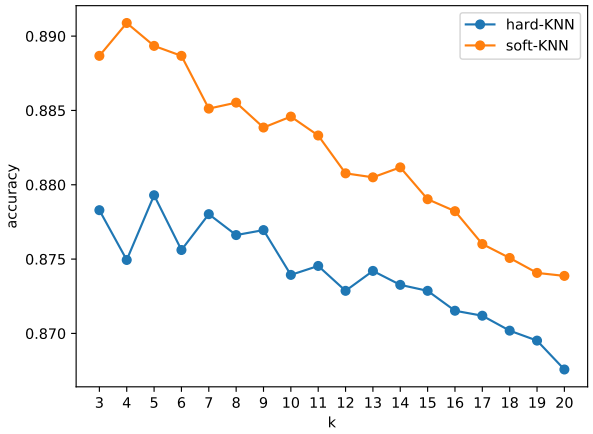

Figure 5. Results of Manhattan Distance with Different ks

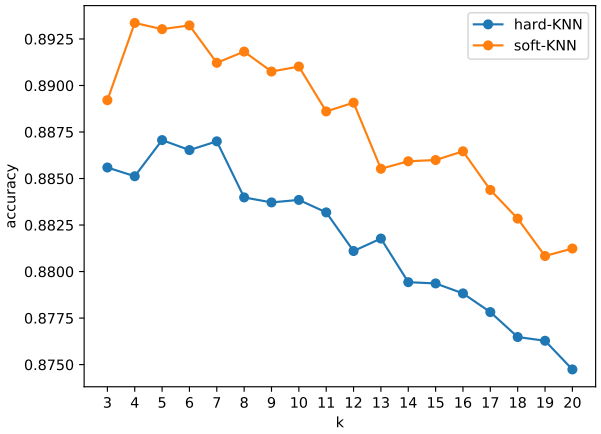

Figure 6. Results of Standard Euclidean Distance with Different ks

all the dimensions of data have the same units.

3.4.4 Standard Euclidean Distance

The only thing Standard Euclidean distance is different from Euclidean distance is that there is a form of standardization in its formula. The relative results are shown in table 4, and the curves of accuracy with different ks are shown in fig. 6.

For hard-KNN, when k is set to 5, we get the highest $\text{accuracy} = 0.88707$; for soft-KNN, when k is set to 4, we get the highest $\text{accuracy} = 0.89336$.

Concluded from the experimental results, Standard Euclidean distance outperforms many other metrics we have experimented with. However, compared to the baseline metric, the accuracy of Standard Euclidean distance is slightly lower, which means there is no benefit to standardize the original Euclidean distance in this case.

Consider the case that some dimensions of the data are on different scales of measurement and the inter-sample differences differ a lot, some form of standardization might be necessary to balance out the contributions, where Standard Euclidean distance probably would help to improve the results.

3.4.5 Hamming Distance

Hamming distance is frequently used in the information theory. The Hamming distance between two vectors of equal length is defined by the number of dimensions at which the the two vectors differ. It is more often used in coding theory. The relative results are shown in table 5, and and the curves of accuracy with different ks are shown in fig. 7.

Table 5. The Results of Hamming Distance

k	Accuracy of Hard-KNN	Accuracy of Soft-KNN
3	0.29185	0.31670
4	0.30853	0.33177
5	0.31730	0.34128
6	0.32219	0.34718
7	0.32440	0.35126
8	0.32862	0.35240
9	0.32989	0.35408
10	0.32822	0.35307
11	0.32802	0.35495
12	0.33023	0.35655
13	0.32949	0.35796
14	0.32969	0.35702
15	0.33090	0.35789
16	0.33217	**0.35816**
17	**0.33358**	0.35676
18	0.33552	0.35716
19	0.33532	0.35689
20	0.33505	0.35615

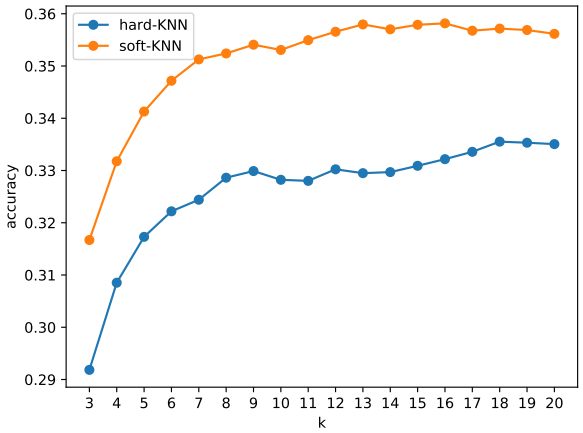

Figure 7. Results of Hamming Distance with Different ks

For hard-KNN, when k is set to 17, we get the highest $\text{accuracy} = 0.33358$; for soft-KNN, when k is set to 16, we get the highest $\text{accuracy} = 0.35816$.

Hamming distance shows poorest performance, and the main reason is that Hamming distance is used for sequences with binary bits. In this experiment, every bit of pre-extracted feature is float, and simply comparing the corresponding bits of two features will cause huge information loss.

3.4.6 Cosine Distance

Cosine distance is predicated on the idea that the difference of two vectors could be better measured by the cosine of the angle between them in stead of the distance between them. It is particularly used in circumstances where the outcome is neatly bounded in $[0, 1]$. The relative results are shown in table 6, and the curves of accuracy with different ks are shown in fig. 8

Table 6. The Results of Cosine Distance

k	Accuracy of Hard-KNN	Accuracy of Soft-KNN
3	0.89300	0.91300
4	0.89400	0.91100
5	0.89700	0.91200
6	0.89700	0.91500
7	0.89900	0.91800
8	**0.90000**	0.91900
9	0.89900	0.91600
10	0.89700	0.91900
11	0.89700	0.91600
12	0.89700	0.91800
13	0.89700	0.92000
14	0.89700	0.91800
15	0.89700	0.91600
16	0.89500	0.91400
17	0.89800	0.91800
18	0.89700	0.92000
19	0.89600	**0.92200**
20	0.89600	0.91800

For hard-KNN, when k is set to 8, we get the highest $\text{accuracy} = 0.900$; for soft-KNN, when k is set to 19, we get the highest $\text{accuracy} = 0.922$.

Compared with the previous methods, Cosine distance performs best, as cosine distance mainly explores the similarity between samples, which is a essential component of image classification.

3.4.7 Canberra Distance

Canberra distance is a version of weighted Manhattan distance, and it has been used as a metric for comparing ranked lists. The relative results is shown in table 7, and the curves of results with different ks are shown in fig. 9.

For hard-KNN, when k is set to 11, we get the highest $\text{accuracy} = 0.90026$; for soft-KNN, when k is set to 12, we get the highest $\text{accuracy} = 0.90334$.

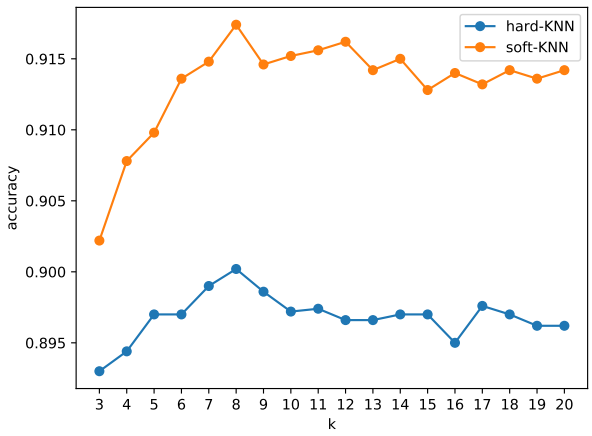

Figure 8. Results of Cosine Distance with Different ks

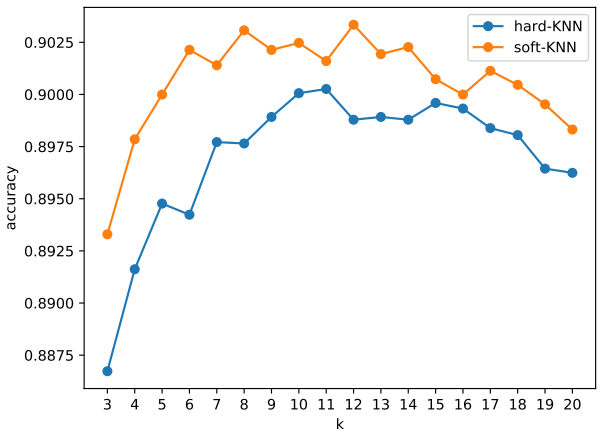

Figure 9. Results of Canberra Distance with Different ks

Table 7. The Results Of Canberra Distance

k	Accuracy of Hard-KNN	Accuracy of Soft-KNN
3	0.88673	0.89329
4	0.89162	0.89785
5	0.89477	0.89999
6	0.89423	0.88674
7	0.89772	0.90140
8	0.89765	0.90307
9	0.89892	0.90214
10	0.90006	0.90247
11	**0.90026**	0.90160
12	0.89879	**0.90334**
13	0.89892	0.90194
14	0.89879	0.90227
15	0.89959	0.90073
16	0.89932	0.89999
17	0.89839	0.90113
18	0.89805	0.90046
19	0.89644	0.89952
20	0.89624	0.89832

Canberra distance is sensitive to small changes when all of the bits are close to zero. In the experiment, we observe that each bit of features is close to zero, which makes Canberra distance reaches relatively better performance than the baseline.

3.4.8 Braycurtis Distance

Braycurtis distance is commonly used in botany, ecology and environmental science field. The relative results is shown in table 8, and the curves of results with different ks are shown in fig. 10.

For hard-KNN, when k is set to 12, we get the highest accuracy $= 0.91131$; for soft-KNN, when k is set to 7, we get the highest accuracy $= 0.91366$.

Table 8. The Results Of Braycurtis Distance

k	Accuracy of Hard-KNN	Accuracy of Soft-KNN
3	0.90274	0.90408
4	0.90240	0.91011
5	0.90689	0.91172
6	0.90716	**0.91366**
7	0.90930	0.91332
8	0.91004	0.91339
9	0.91084	0.91346
10	0.91038	0.91245
11	0.91038	0.91125
12	**0.91131**	0.91218
13	0.91118	0.91105
14	0.90997	0.91138
15	0.91064	0.90971
16	0.91038	0.91051
17	0.90870	0.90957
18	0.90991	0.91038
19	0.90870	0.90870
20	0.90950	0.90910

Braycurtis distance is very similar to Canberra distance, and it reaches much better performance than Canberra distance. We can easily observe that for two arbitrary features $\mathbf{x}$ and $\mathbf{y}$, $d_{canberra}(\mathbf{x}, \mathbf{y}) \geq d_{braycurtis}(\mathbf{x}, \mathbf{y})$. Though Canberra distance will make the features more discrete, its accuracy is still lower than Braycurtis distance, which means Braycurtis distance is more consistent with the distribution of this dataset.

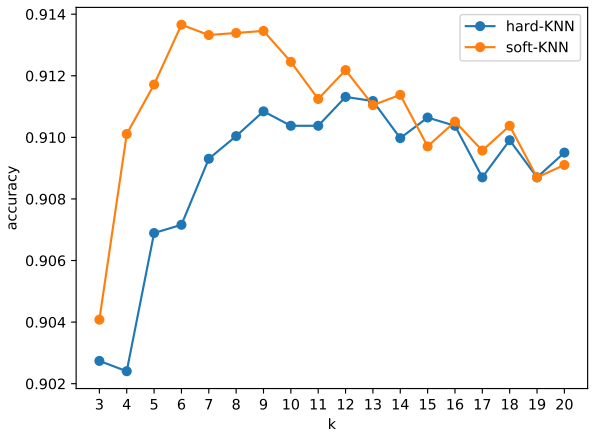

Figure 10. Results of Braycurtis Distance with Different ks

3.5. Results of Metric Learning Methods

Let $\mathbf{X}$ represents the matrix of samples. Different metric learning methods transform $\mathbf{X}$ into different $\mathbf{X}'$. For better comparison, we first built soft-KNN euclidean model with $k = 5$, then load original sample matrix $\mathbf{X}$ and converted $\mathbf{X}'$, by which, we got whether the transformed matrix is better and how good it is.

3.5.1 Baseline

Without any metric learn method, we only use original matrix $\mathbf{X}$ and we get the $\text{accuracy} = 0.89751$.

3.5.2 NCA

We get the $\text{accuracy} = 0.91486$, highest among our listed metric learning method. The learning strategies of NCA is to extend the nearest neighbor classifier toward metric learing, proposed in learns a Mahalanobis distance metric for the KNN classifier by maximizing the leave-one-out cross validation.

Actually NCA tends to overfit the training data if the number of training examples is insufficient for learning a distance metric. This happens often when data points are represented in the high dimensional space. However, our experimental result shows that NCA fits well. It's probably due to the reason that we have enough amount of data.

3.5.3 LMNN

We get the $\text{accuracy} = 0.90676$, a little bit higher than baseline. Because Large Margin Nearest Neighbor method learns a Mahanalobis distance metric in the KNN-classification setting using semidefinite programming, Learned metric attempts to keep k-nearest neighbors in the same label, while keeping examples from different classes separated by a large margin. Our model is also KNN, which means that LMNN and KNN are highly coupled.

Meanwhile, This algorithm makes no assumptions about the distribution of the data, which makes the generalization of the algorithm is improved.

3.5.4 MLKR

We get the $\text{accuracy} = 0.89651$, The main differences between NCA and MLKR are that MLKR minimizes a regression error loss function, whereas NCA is designed for classification, and that NCAs probabilistic formulation assumes a specific soft-max probability model.

In the beginning, Kernel regression is a well-established method for nonlinear regression in which the target value for a test point is estimated using a weighted average of the surrounding training samples. Further, MLKR can be viewde as variation of PCA and can be used for dimensionality reduction.

Therefore, the reason why the accuracy becomes lower than baseline is probably because that it's not general enough for classification problems. And compared with NCA which performs the best, the loss function of MLKR is not good as NCA's.

3.5.5 Comparison between Different Metric Learning Methods

The running time of metric learning is pretty long, and with limited due time and computational resources we cannot get the accuracy of different ks. However, the good news is that the experimental result of different distance metric shows that accuracy gap between $k = 5$ and the optimal k is very small. The accuracy of different metric learning methods is shown in table 9.

Table 9. Accuracy of Different Metric Learning Methods ($k = 5$)

Method	Accuracy
Baseline (No metric learning)	0.89751
LMNN	0.90676
NCA	**0.91486**
MLKR	0.89651

4. Discussion

In this section, we discuss the performance of different distance metric related methods, and then give some our observations and the final conclusion of which method is the best choice.

4.1. Comparison between Different Distance Metrics

We ranked all the distance metrics according to their optimal accuracy and the results are shown in table 10.

Table 10. Final **Optimal Accuracy** Rank Among All Listed Distance Metrics

Rank	Distance Metric	Accuracy	Settings
1	Cosine	**0.922**[1]	S[2], $k = 19$
2	Brayburtis	0.91366	S, $k = 6$
3	Canberra	0.90334	S, $k = 12$
4	Euclidean	0.89705	H, $k = 9$
5	Seuclidean[3]	0.89336	S, $k = 4$
6	Manhattan	0.89088	S, $k = 4$
7	Chebyshev	0.79476	S, $k = 6$
8	Hammming	0.35816	S, $k = 16$

[1] The running time of cosine distance metric is pretty long, and with limited due time and computational resources we cannot get a complete prediction. We sampled 5000 (about 35% of total prediction set) to get the result, which is also the reason the accuracy is only three decimal places.
[2] For experiment settings, S represents the soft-KNN algorithm while H represents the hard-KNN algorithm.
[3] Seuclidean is the short for Standard Euclidean

According to table 10, we have some observations:

- The selection of k is depend on which distance metric we are using, which suggests that different distance metric will make the distribution of data different.
- For almost all of the distance metrics (except Euclidean distance), soft-KNN outperforms hard-KNN.
- Soft-KNN using Cosine distance reaches the best performance and $\text{accuracy} = 0.922$.

4.2. Comparison between Simple Distance Metrics and Metric Learning

Then we compare the simple distances metric which reaches the best performance among all of the distance metrics and the the metric learning method which reaches the best performance among all of the metric learning methods. The results is shown in table 11.

Table 11. Accuracy of Best Simple Distance Metric and Best Metric Learning Method

Method	Accuracy
Soft-KNN + Cosine	**0.922**
NCA	0.91486

According to table 11, soft-KNN with Cosine distance reaches the best performance among all of the distance metric related methods. Due to the limitation of time and the huge temporal overhead of metric learning methods, we just test their performance when $k = 5$ and maybe they will have better performance if we test more ks. However, we can still see that soft-KNN with Cosine distance is much more efficient than metric learning methods and it can get promising classification results.

Thus, we can draw a conclusion that soft-KNN with Cosine distance is the best choice of distance metric related methods considering computational overhead and classification performance.

5. Acknowledgement

Thank teammates for their cooperation and thank TAs and Prof. Li Niu for their guidance.

References

[1] Importance of distance metrics in machine learning modelling. https://en.wikipedia.org/wiki/Metric_(mathematics).

[2] metric-learn: Metric learning in python. http://metric-learn.github.io/metric-learn/.

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

[4] R. Jin. Distance metric learning : A comprehensive survey. 2006.

[5] J. Suárez, S. García, and F. Herrera. A tutorial on distance metric learning: Mathematical foundations, algorithms and software. *CoRR*, abs/1812.05944, 2018.

[6] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1473–1480. MIT Press, 2006.

[7] K. Q. Weinberger and G. Tesauro. Metric learning for kernel regression. In M. Meila and X. Shen, editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 612–619, San Juan, Puerto Rico, 21–24 Mar 2007. PMLR.