

K-Nearest Neighbors(KNN) Classification with Different Distance Metrics

Project 2 for Principle of Data Science (CS245)

Zijie Zhu (朱子杰) 517030910389

Mengtian Zhang (张孟天) 517030910387

May 11, 2020

Abstract

In this experiment, we explore the basic principle of KNN Algorithm in detail, as well as the calculation formulas of various distance metrics it uses. We studied several algorithms of metric learning to improve the performance of classification model. We have carried out many experiments with AwA2 as the dataset. Under the test of various experimental parameters, the applicability of various distance measures is concluded.

1 Introduction

In the daily research of artificial intelligence, it is often necessary to quantify various unstructured data. The image, audio, text and other data are transformed into numerical vector form for storage, or the vector form features are extracted to represent the original data. In this way, not only the data can be saved conveniently, but also the similarity between unstructured data can be quantified by the distance between vectors.

In the task of classification, KNN is an algorithm that uses distance metrics to classify different samples. Intuitively, the two samples with small distance tend to have higher similarity, so they are more likely to belong to the same label. KNN can use this feature to observe the neighborhood around a single sample and classify the samples. However, there are many details that affect the classification results. These different settings will be tried in the following experiments.

As a traditional learning method, KNN is carried out without changing the spatial distribution of samples. However, this method is easy to cause classification errors due to outliers, which affects the accuracy. Therefore, we can use metric learning to make classification more error free. The general idea of doing this is to learn the projection matrix to increase the distance between different sample points and reduce the distance between the same sample points. In the next experiment, we also experiment with different metric learning algorithms.

2 Method Analysis

2.1 KNN

KNN is classified by measuring the distance between different eigenvalues. Its idea is: if most of the k most similar samples in the feature space belong to a certain category, then the sample also belongs to this category, where k is usually an integer no more than 20. In KNN algorithm, the selected neighbors are all correctly classified objects. In the decision-making of classification, this method only depends on the category of the nearest sample or samples to determine the category of the samples to be classified.

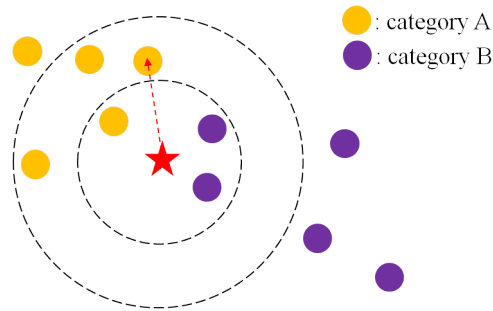


Figure 1: Illustration of KNN Algorithm

We can further describe the algorithm as:

- Calculate the distance between the test data and each training data;
- Order according to the increasing distance;
- Select k points with the smallest distance;
- Determine the occurrence frequency of the category of the first k points;
- Return the most frequent category in the first k points as the prediction classification of test data.

Therefore, the function used to measure distance and the selection of K play a great role in the results.

2.2 Different Distance Metric Method

2.2.1 Euclidean Distance

Euclidean distance can be said to be the distance metric most in line with our daily intuition. It reflects the linear distance between two points in Euclidean space. The formula for calculating Euclidean distance is:

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

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

2.2.2 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.

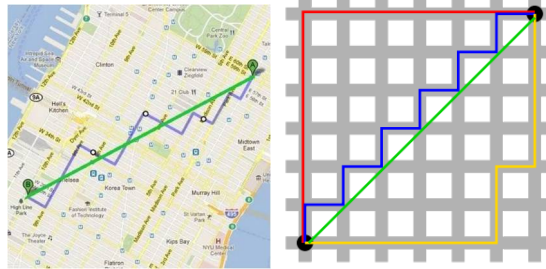


Figure 2: the Manhattan Distance

The formula for calculating Manhattan distance is:

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

2.2.3 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. Because Wang can move one grid in the oblique forward or backward direction, he can reach the target grid more efficiently.


	a	b	c	d	e	f	g	h	
8	5	4	3	2	2	2	2	2	8
7	5	4	3	2	1	1	1	2	7
6	5	4	3	2	1		1	2	6
5	5	4	3	2	1	1	1	2	5
4	5	4	3	2	2	2	2	2	4
3	5	4	3	3	3	3	3	3	3
2	5	4	4	4	4	4	4	4	2
1	5	5	5	5	5	5	5	5	1
	a	b	c	d	e	f	g	h	

Figure 3: Chebyshev Distance in Chess

2.2.4 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, corresponding to L^1 norm, L^2 norm and L^∞ norm respectively.

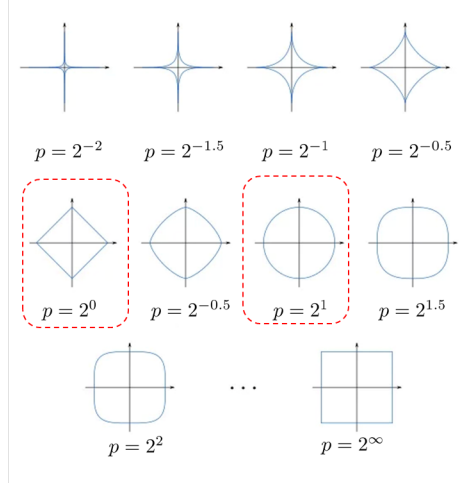


Figure 4: A Sketch of Minkowski Distance under Different P-values

2.2.5 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:

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

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

$$d_{euc}(\mathbf{x}, \mathbf{y}) = \sqrt{2 - 2Sim_{cos}(\mathbf{x}, \mathbf{y})}$$

2.2.6 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 $x = (x_1, x_2, \dots, x_p)^T$ whose mean value is $\mu = (\mu_1, \mu_2, \dots, \mu_p)^T$ and covariance matrix is Σ , its Mahalanobis distance is:

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

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 :

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

2.3 Metric Learning

When the two samples are heterogeneous, the above distance metric method is no longer applicable. It is necessary to propose a new method to project two kinds of different samples into the same space.

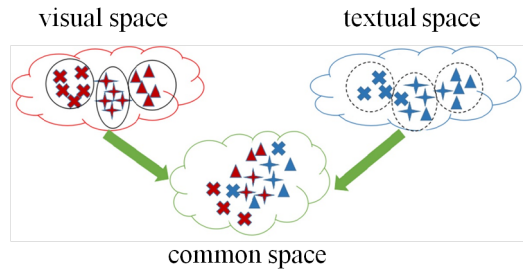


Figure 5: Project Heterogeneous Data into the Same Space

Here comes metric learning. In metric learning, we no longer consider the distance between X and Y, instead, they are multiplied by the same projection matrix P. We then calculate the distance as follows:

$$d(Px, Py) = \|Px - Py\|_2 = \sqrt{(x - y)^T P^T P (x - y)}$$

In practical application, we record $P^T P$ together as M. M is what we want to learn and if M is unit matrix, it is equivalent to Euclidean distance.

2.3.1 LMNN

Large margin nearest neighbor (LMNN) classification is a statistical machine learning algorithm for metric learning. It learns a pseudometric designed for k-nearest neighbor classification. The algorithm is based on semidefinite programming, a sub-class of convex optimization.

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:

$$\min_{\mathbf{L}} \sum_{i,j} \eta_{ij} \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)\|^2 + c \sum_{i,j,l} \eta_{ij} (1 - y_{ij}) [1 + \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)\|^2 - \|\mathbf{L}(\mathbf{x}_i - \mathbf{x}_l)\|^2]_+$$

where \mathbf{x}_i is an 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 class, $[\cdot]_+ = \max(0, \cdot)$ is the Hinge loss.

2.3.2 NCA

Neighborhood Components Analysis (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 $\mathbf{M} = \mathbf{L}^T \mathbf{L}$ and define the probability p_{ij} that \mathbf{x}_i is the neighbor of \mathbf{x}_j by calculating the softmax likelihood of the Mahalanobis distance:

$$p_{ij} = \frac{\exp(-\|\mathbf{L}\mathbf{x}_i - \mathbf{L}\mathbf{x}_j\|_2^2)}{\sum_{l \neq i} \exp(-\|\mathbf{L}\mathbf{x}_i - \mathbf{L}\mathbf{x}_l\|_2^2)}, \quad p_{ii} = 0 \quad (1)$$

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

$$p_i = \sum_{j: j \neq i, y_j = y_i} p_{ij} \quad (2)$$

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

$$\mathbf{L} = \arg \max \sum_i p_i \quad (3)$$

2.3.3 LSML

Least Squares Metric Learning (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 $d(\mathbf{x}_a, \mathbf{x}_b) < d(\mathbf{x}_c, \mathbf{x}_d)$ is denoted as:

$$H(d_{\mathbf{M}}(\mathbf{x}_a, \mathbf{x}_b) - d_{\mathbf{M}}(\mathbf{x}_c, \mathbf{x}_d)) \quad (4)$$

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

$$H(x) = \begin{cases} 0 & x \leq 0 \\ x^2 & x > 0 \end{cases} \quad (5)$$

The summed loss function $L(C)$ is the simple sum over all constraints $C = \{(\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d) : d(\mathbf{x}_a, \mathbf{x}_b) < d(\mathbf{x}_c, \mathbf{x}_d)\}$, The original paper suggested here should be a weighted sum since the confidence or probability of each constraint might differ. However, for the sake of simplicity and assumption of no extra knowledge provided, we just deploy the simple sum here as well as what the authors did in the experiments.

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

$$\min_M D_{ld}(\mathbf{M}, \mathbf{M}_0) + \sum_{(\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d) \in C} H(d_{\mathbf{M}}(\mathbf{x}_a, \mathbf{x}_b) - d_{\mathbf{M}}(\mathbf{x}_c, \mathbf{x}_d)) \quad (6)$$

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

$$D_{ld}(\mathbf{M}, \mathbf{M}_0) = tr(\mathbf{M}\mathbf{M}_0^{-1}) - \log \det(\mathbf{M}) \quad (7)$$

3 Experiment

3.1 Dataset

The dataset we use in this experiment is AwA2 containing 37322 images of 50 animal classes with pre-extracted deep learning features for each image. Each image is represented in a 2048-dimension feature vector.

In this experiment, we split the dataset as train set with 60% quantity of the images and 40% as test set. Use `train_test_split()` function in `sklearn` can easily complete this target.

3.2 Function Prototype of Main Application

In the basic KNN experiment, the main function we used is `KNeighborsClassifier()` in `sklearn` library. The prototype of `sklearn.neighbors.KNeighborsClassifier` is shown as follows:

```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *,
weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski',
metric_params=None, n_jobs=None, **kwargs)
```

As for the basic KNN experiment, we'll take this function as the main object, and carry out experiments on different settings of its different parameters, in order to observe its influence on the experimental results. Next, we will briefly introduce the parameters we studied one by one.

- **n_neighbors** Corresponding to the K-value in KNN algorithm. That is, the number of neighbors to use by default for k-neighbors queries.
- **weights** Weight function used in prediction, defining the representation of different distances from multiple points to query points. Possible values are: `'uniform'` and `'distance'`.

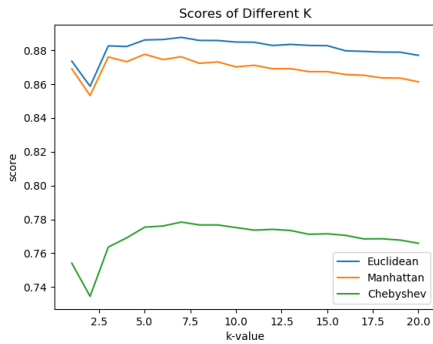


Figure 6: Different k in Different Metrics

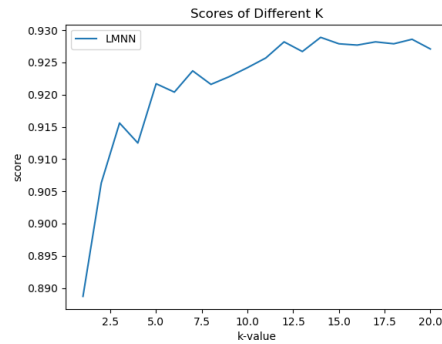


Figure 7: Different k in LMNN

- **algorithm** Algorithm used to compute the nearest neighbors. Different algorithms will affect the calculation time of training, but will not affect the final accuracy. Possible values are *'auto'*, *'ball_tree'*, *'kd_tree'* and *'brute'*.
- **leaf_size** Passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree.
- **p** Power parameter for the Minkowski metric.
- **metric** The distance metric to use for the tree. Possible values are: *'euclidean'*, *'chebyshev'*, *'manhattan'* and *'minkowski'*.

3.3 Testing on Different K-Values and Distance Metrics

As a hyper-parameter, we use k-fold algorithm to test different K values in order not to affect the experimental results on the test set. At the same time, we test the different distance metrics, and get the most appropriate K value for each metric.

We split the train set into 4 folds and calculate the average value of four scores on every fold. The candidate k-value is from 1 to 20 and the metrics include Euclidean distance, Manhattan distance and Chebyshev distance. Their score is list in table 1.

It can be concluded that the best k-value is 7 for Euclidean distance, 5 for Manhattan distance and 7 for Chebyshev distance. In general, the accuracy score increases first and then decreases with the increase of K value. And the result of Euclidean distance and Manhattan distance is much better than Chebyshev distance.

An interesting phenomenon is that when $k = 2$, the scores of the three distance metrics all have a pool performance, i.e. the score is lower than when $k = 1$ or 3. An intuitive explanation is related to KNN's majority voting mechanism. When $k = 2$, once the other samples closest to the inquiry point belong to two categories, the judgment result will be confused. This phenomenon also exists in other cases when k is small and even. Therefore, in KNN experiments, it is often better to take the odd number of K.

In most of the next experiment, we take the Euclidean distance of $k = 7$ as the baseline.

Table 1: Scores of Different K-Values and Distance Metrics

k \ metrics	Euclidean	Manhattan	Chebyshev
1	0.873574928	0.869067779	0.754082744
2	0.858746833	0.853251560	0.734469576
3	0.882640643	0.876121099	0.763634164
4	0.882233810	0.873261114	0.769033672
5	0.886166645	0.877681487	0.775419154
6	0.886387524	0.874552529	0.776092429
7	0.887682678	0.876205315	0.778411189
8	0.885851826	0.872319899	0.776708430
9	0.885807020	0.873169922	0.776712046
10	0.884914353	0.870221040	0.775148906
11	0.884778523	0.871157668	0.773631542
12	0.882858633	0.869104165	0.774119667
13	0.883529908	0.869151189	0.773404603
14	0.882904215	0.867410079	0.771173847
15	0.882769518	0.867407525	0.771484988
16	0.879733413	0.865711665	0.770551230
17	0.879373540	0.865262810	0.768454285
18	0.878974370	0.863699119	0.768542590
19	0.878886045	0.863565953	0.767739959
20	0.877052482	0.861377776	0.765866284

3.4 Testing on Different P-values in Minkowski Distance

Next, we do experiments to explore the effect of the value of P in Minkowski distance on the results. In the diagram, we can see that Minkowski distance can be regarded as the norm of vector difference. And the larger the p value is, the fuller the envelope of norm in vector space is. Under the condition of $k = 7$, we do experiments on P from 1 to 10. The results of the training model on the test set are listed in table 2.

Table 2: Scores of Different P-values in Minkowski Distance

P Value	1	2	3	4	5	6	7	8	9	10
Accuracy	0.8816	0.8938	0.8823	0.8677	0.8559	0.8437	0.8351	0.8297	0.8234	0.8195

The above results have four decimal places reserved. The result is that $p=2$, i.e. Euclidean distance is the best choice for Minkowski distance in this experiment. From then on, the accuracy is lower and lower with p-value increases. Since Manhattan distance is equivalent to Minkowski distance with $p=1$ and Chebishev distance equivalent to $p=\infty$. The conclusion that Euclidean distance is slightly better than Manhattan distance and both are much better than Chebyshev distance is verified again.

3.5 Influence of Data Normalization

As a widely used data preprocessing method, normalization can improve the training results. Especially for the data with different dimensions of heterogeneous content, normalization can balance the contribution of each dimension to classification calculation and eliminate the possibility of excessive influence of one dimension on the results.

In this experiment, we try to normalize the data with Z-score before training, and compare the final result with baseline. In addition, we also tried min-max standardization, and listed it in the table 4, too. We select Euclidean distance with $k=7$, Manhattan distance with $k=5$ and Chebyshev distance with $k=7$ as the experiment parameter.

Table 3: Scores after Normalization

metrics	Euclidean	Manhattan	Chebyshev
Z-Score	0.8797	0.8710	0.6938
Min-Max	0.8926	0.8819	0.7620
Baseline	0.8938	0.8827	0.7853

From the results, we can see that after normalization, the accuracy of training has decreased a lot, which is unexpected. Even standardization makes the results slightly worse. After careful consideration, we conclude that the main role of normalization is to prevent different contributions of different dimensions to classification, especially for heterogeneous features.

However, the features extracted in awa2 are likely to be key information after preprocessing rather than heterogeneous. At this time, normalization will make the feature lose a lot of information available for classification. And the normalized data is more dense, which is not conducive to the separation of different types of samples.

3.6 Testing on Different Algorithm Used to Compute

In the function, "*algorithm*" is an alternative parameter which use different algorithms to compute the nearest neighbors.

Table 4: The score and time of different algorithms

Algorithm	ball_tree	kd_tree	brute	auto
Score	0.8942	0.8942	0.8942	0.8942
Time(s)	1168.93	1555.50	35.81	1558.97

- "*ball_tree*": will use BallTree (a space partitioning data structure for organizing points in a multi-dimensional space).
- "*kd_tree*": will use KDTree (a space-partitioning data structure for organizing points in a k-dimensional space).

- *"brute"*: will use a brute-force search.
- *"auto"*: will attempt to decide the most appropriate algorithm based on the values passed to fit method.

And to analysis the effect of different algorithms, we use euclidean distance and set nearest neighbors $k = 7$.

From Table 4, we can find that different algorithms will have a huge difference in running time but get the same final score. When we use 'auto' parameter, the system might automatically selects 'kd_tree' as the default algorithm. And we should select appropriate algorithm for different datasets.

3.7 Testing on Different Choices of Distance Weight

As a parameter in the function, we hope to do an experiment for different weights. Different parameters are interpreted as:

- *'uniform'*: uniform weights. All points in each neighborhood are weighted equally.
- *'distance'*: 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.

Considering the effect of distance weight on different distance metrics may cause different performance, we select Euclidean distance with $k=7$, Manhattan distance with $k=5$ and Chebyshev distance with $k=7$ as the experiment parameter. The result is shown in table 5:

Table 5: Scores of Different Weight

weights	Euclidean	Manhattan	Chebyshev
uniform	0.8938	0.8816	0.7853
distance	0.8959	0.8867	0.7928

The results show that for different distance measures, using weighted distance calculation can achieve better results. Guess in the case of dense data points, the weighted calculation can make the closer neighbors receive more attention, thus reducing the possibility of interference of other categories of points.

3.8 Metric Learning

all distance metric learning algorithms aims to optimize the matrix \mathbf{M} in Mahalanobis distance. And in this lab, we try several metric learning algorithms provided in *"metric-learn"* package, including Large Margin Nearest Neighbor (LMNN), Neighborhood Components Analysis (NCA), Least Squares Metric Learning (LSML) etc. And the experiment comparison result is shown in Table 6.

Baseline. In the baseline part, we do not do any metric learning algorithm (In other words, set $\mathbf{M} = I$). That is, we just use the raw feature vectors and Euclidean distance to do the KNN classifier. And so the running time in baseline is just the running time for KNN classifier once. That is, the time in other methods minus the baseline time will be the metric learning time.

LMNN. In the *LMNN()* function, we set the max iterations as 200. And in this part, we set k (the number of neighbors to consider) as 7. In the next part, we will also have a further study of the value of k.

NCA. In the *NCA()* function, we also set the max number of iterations as 200.

LSML. In the *LSML()* function, we use the "identity" matrix as the prior and set the max number of iterations as 200.

Table 6: Different metric learning methods

Method	Baseline	LMNN	NCA	LSML
Score	0.8877	0.9237	0.9044	0.9056
Time(s)	40.81	4748.36	17137.97	1821.73

From Table 6, we can clearly get that,

- All metric learning methods can improve the final accuracy score compared with the benchmark.
- Also due to the large number of iterations, metric learning algorithms takes a lot of time to converge.
- In the three tested methods, LMNN has the best score result.
- In the three tested methods, LSML takes the least time.

Metric learning methods use the Mahalanobis Distance and try to optimize the matrix **M**. And so they can improve classifier accuracy significantly. But in the process of metric learning experiment, we also find some weakness. that is, all the metric learning algorithms have large time and space complexity, due to complex matrices operations and large number of iterations. To reduce the running time and memory occupation, we can use some dimensionality reduction algorithms or select a subset of dataset randomly to do the metric learning process. For example, I try to use *pca* to reduce the features to 1500 dimensions first in NCA and then it still has a similar accuracy and a lower running time.

3.9 Testing on Different K-values in LMNN

Pylmnn is a LMNN implementation library specially applied to Python environment. The *LargeMarginNearestNeighbor* class can be used to complete the experiment task conveniently. Its function prototype is:

```
class pylmnn.lmnn.LargeMarginNearestNeighbor(n_neighbors=3, n_components=None,
init='pca', warm_start=False, max_impostors=500000, neighbors_params=None,
weight_push_loss=0.5, impostor_store='auto', max_iter=50, tol=1e-05,
callback=None, store_opt_result=False, verbose=0, random_state=None, n_jobs=1)
```

In order to further explore the characteristics of lmn algorithm, we experiment the key super parameter K in the model. We use different k-value in LMNN model and set the projection result to the same 2048 dimension as the original data. And perform KNN classification again at the Euclidean distance of $k = 7$. Due to the limitation of training time, each model has undergone 30 iterations. Although this will make the training results difficult to converge. But in order to get the trend of the result with K, it is already enough.

The test results under different LMNN-k values are listed in table 5.

Table 7: Scores of Different k-values in LMNN

K Value	1	2	3	4	5	6	7	8	9	10
Accuracy	0.8887	0.9063	0.9156	0.9125	0.9217	0.9204	0.9237	0.9216	0.9228	0.9242
K Value	11	12	13	14	15	16	17	18	19	20
Accuracy	0.9257	0.9282	0.9267	0.9289	0.9279	0.9277	0.9282	0.9279	0.9286	0.9271

In general, the higher the K value of lmn model, the higher the accuracy score. The final fraction tends to converge. Since the k controls how many other samples around the inquiry point are the largest margin of the same category, it is possible to guess that the training requirements are more strict when the k-value is larger. And therefore make it better on the test set. The visualized figure of score variation is shown in figure 7.

4 Summary

In this project, we learn to use the k-nearest neighbors (KNN) classification with different distance metrics. And Figure 8 shows the classification accuracy for different distance metrics. For traditional simple distance metrics (Chebyshev distance, Euclidean distance, Manhattan distance), Euclidean distance achieve better performance. Metric learning algorithms for Mahalanobis distance can improve the classifier accuracy and among them LMNN can get the best performance.

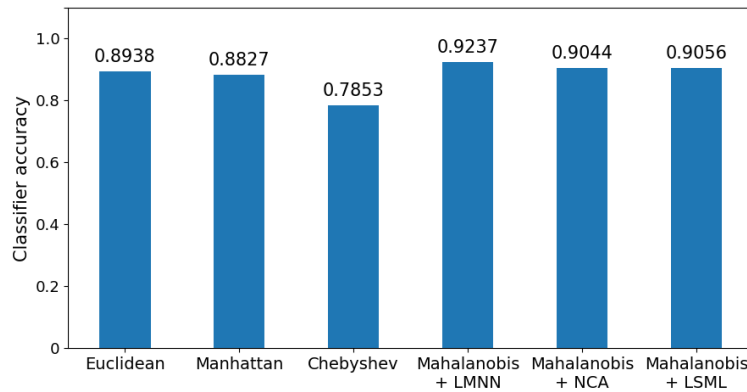


Figure 8: Classification accuracy for different distance metrics

And next I will summarize all the work that we've done.

Firstly, we use the K-fold cross-validation within the training set to determine the value of K in KNN for different distance metrics. And the **best k-value** is **7** for Euclidean distance, **5** for Manhattan distance and **7** for Chebyshev distance.

Next, we try to use different p-values in minkowski distance and observe the classifier results. And The conclusion is that **p = 2**, i.e. Euclidean distance, is the best choice for Minkowski distance in this experiment.

And we also study the influence of data normalization (z-score and max-min) in data preprocessing part, the algorithm selection in KNN, and the distance weight selection in KNN.

Finally, we do lots of work to study metric learning method. And we try to use several metric learning methods available (LMNN, NCA, LSML) to learn good metrics, that is good **M** in Mahalanobis distance. And the experimental results show that metric learning methods can improve the classification accuracy significantly. But also metric learning algorithms need a large amount of time and memory (space).