ARIEL UNIVERSITY OF SAMARIA

MASTERS THESIS

Multidimensional Interpolated Discretized for Objects and Object Pairs Embedding

Author:
Yakir BEN-ALIZ

Supervisor:
Dr. Ofir PELE

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science

in the

Faculty of Engineering
electrical engineering

June 26, 2017

ARIEL UNIVERSITY OF SAMARIA

Abstract

Faculty of Engineering
electrical engineering

Master of Science

Multidimensional Interpolated Discretized for Objects and Object Pairs Embedding

by Yakir BEN-ALIZ

Distance functions are at the core of numerous scientific areas. One can choose a distance function based on prior knowledge or learn it from data, metric learning. The most commonly used and learned distance function is the Euclidean distance. In metric learning, most of the works learn a Mahalanobis distance. These methods [1-12] learn a linear transform that is applied on the vector and then apply the squared Euclidean distance (thus these methods are actually semimetric learning). Kernel metric learning applies embedding separably on each vector before learning the linear transform. Deep learning methods [13] learn an embedding using a deep network and then apply the Euclidean distance on the embedded vectors (the output of the network). Thus, even kernel and deep metric learning can only learn a Euclidean distance. Some works [14, 15, 16, 17] have suggested learning other families of distances. However, these methods are restricted to the suggested pre-chosen families of distances (e.g. Earth Mover's Distance and χ Finally, multi-metric learning methods [1, 18] learn separate local Mahalanobis metrics around keypoints. However, they do not learn a global metric. An exception is [19] which shows how to combine information from several local metrics into one global metric. However, again it is only able to model Euclidean metrics.

We propose a new embedding method for a single vector and for a pair of vectors. This embedding method enables:

- efficient classification and regression of functions of single vectors
- efficient approximation of distance functions
- general, non-Euclidean, semimetric learning

To the best of our prior knowledge, this is the first work that enables learning any general, non-Euclidean, semimetrics. That is, our method is a universal semimetric learning and approximation method that can approximate any distance function with as high accuracy and/or without semimetric constraints. The main difference between our model and previous models is that our model embeds object pairs jointly and not separably. Distance between objects is the embedded vector dot product with a learned parameters vector. Thus, most of the learning objectives are convex in our model. Additionally, we can enforce constraints on the vector of parameters such that the resulting distance will be a continuous semimetric. This work enables learning and approximation of arbitrary distance functions or arbitrary semimetrics.

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ID	Interpolized Discretized
NDIDD	n-Dimension Interpolized Discretized Distance
BCC	Bary centric coordinates
SGD	Stochastic Gradient Descent

/Dedicated to/To my…

Chapter 1

Introduction

Distance functions are at the core of numerous scientific areas , such as classification, regression, clustering challenges etc. it can be based either on strict, constant formulation, such as norms [23] (such as L_2 norm, representing the euclidean distance [4]), or can be learned from datasets - metric learning.

In metric learning, most of the works learn a Mahalanobis distance [5]. These methods learn a linear transformation that is applied on the vector and then apply the squared Euclidean distance (thus these methods are actually semimetric learning). Kernel metric learning applies embedding separably on each vector before learning the linear transformation. Deep learning [10] methods learn an embedding using a deep network and then apply the Euclidean distance on the embedded vectors (the output of the network). Thus, even kernel and deep metric learning can only learn a Euclidean distance. Some works [14, 15, 16, 17] have suggested learning other families of distances. However, these methods are restricted to the suggested pre-chosen families of distances (e.g. Earth Mover's Distance and χ. Finally, multimetric learning methods [1, 18] learn separate local Mahalanobis metrics around keypoints. However, they do not learn a global metric. An exception is [19] which shows how to combine information from several local metrics into one global metric. However, again it is only able to model Euclidean metrics.

Our work is handling the following use case: let us say there is a given dataset, which does not own any euclidean properties, and cannot be embedded separately in order to perform various classification tasks. Ofir's work [14] treats this particular matter, by interpolating and embedding pairs of data as a unified objects, by performing bin to bin semi-metric pairing.

We propose a new embedding method for a single vector and for a pair of vectors. This embedding method enables:

- efficient classification and regression of functions of single vectors
- efficient approximation of distance functions
- general, non-Euclidean, semimetric learning

Bin to bin comparison between pairs of data samples is beneficial when the data is not dimensionally correlated, and has no relations between one dimension in the first vector, to another. For example, when performing SIFT [1] analysis to compare between two images for pattern recognition, there might be relations between one element to its own neighbors, but also to its paired - candidate neighbor. For this and other purposes we may consider a cross-bin comparison method.

Let us now address describing the theories behind our method.

1.1 Metrics

1.1.1 Definition

Metric space is a set for which distances between all members of the set are defined. Distances applied on every pair of objects on a given set called **metric**. A metric d is defined as:

$$d : q_1 \times q_2 \to \Re \tag{1.1}$$

where q_i are objects in a given set

1.1.2 Metrics Properties

Any metrics must obey the following properties:

1.1.2.1 Non-negativity

any metric on a pair of objects must be non-negative

$$d(q_1, q_2) \geq 0 \tag{1.2}$$

1.1.2.2 Identity of Indiscernibles

$$d(q_1, q_2) = 0 \iff q_1 = q_2 \tag{1.3}$$

for every pair of objects q_1, q_2, d metric function provides zero if and only if those objects are identical. Identity of indiscernibles is an ontological principle that states there cannot be separate objects or entities that have all their properties in common.

1.1.2.3 Symmetry

$$d(q_1, q_2) = d(q_2, q_1) \tag{1.4}$$

A symmetric function of a pair of objects is one whose value at any pair of objects is the same as its value at any permutation of that pair.

1.1.2.4 Sub-additivity (Triangle Inequality)

$$d(q_1, q_3) \leq d(q_1, q_2) + d(q_2, q_3) \tag{1.5}$$

Evaluating the function for the sum of two elements of the domain always returns something less than or equal to the sum of the function's values at each element.
There are two useful generalizations for metric definition:

1.1.3 Semi-metrics

Semi metric is a generalization of the metric definition, which basically excludes 1.1.2.4, and remains the rest.

1.1.4 Pseudo-metrics

Pseudometrics supports all metrics properties except the identity of indiscernibles property 1.1.2.2, which is modified as follows:

$$q_1 = q_2 \Rightarrow d(q_1, q_2) = 0 \tag{1.6}$$

1.2 Metric Learning

Metric learning study refers to learning a distance function from data objects, while still applying the basic properties of metrics.

Most of the works learn a Mahalanobis distance[]. These methods [1-12] learn a linear transform that is applied on the vector and then apply the squared Euclidean distance (thus these methods are actually semimetric learning).

Kernel metric learning applies embedding separably on each vector before learning the linear transform.

Deep learning methods such [13] learn an embedding using a deep network and then apply the Euclidean (or any known) distance on the embedded vectors (the output of the network). Thus, even kernel and deep metric learning can only learn a Euclidean distance. Some works [14-17] have suggested learning other families of distances. However, these methods are restricted to the suggested pre-chosen families of distances (e.g. Earth Mover's Distance[] and χ^2[].

Finally, multi-metric learning methods [1, 18] learn separate local Mahalanobis metrics around keypoints. However, they do not learn a global metric. An exception is [19] which shows how to combine information from several local metrics into one global metric. However, again it is only able to model Euclidean metrics.

1.3 Bin-to-Bin & Cross-Bin Metrics

Bin-to-Bin distance functions such as L_2, L_1 and χ^2 compare only corresponding bin's of a vector to its exact corresponding bin in the second vector. The assumption when using these distances is that the histogram domains are aligned. However this assumption is violated in many cases due to quantization, shape deformation, light changes, etc. Bin-to-bin distances depend on the number of bins. If it is low, the distance is robust, but not discriminative, if it is high, the distance is discriminative, but not robust. Distances that take into account cross-bin relationships (cross-bin distances) can be both robust and discriminative.

1.4 Mahalanobis Distance

Let $A \in \Re^{N\times N}$ be a bin-similarity matrix, so that a_ij encodes how much bin i is similar to bin j.

The Quadratic-Form (QF) distance [21] is defined as :

$$QF^A(P, Q) = \sqrt{(P - Q)^T \times A(P - Q)} \tag{1.7}$$

Where the bin-similarity matrix A is the inverse of the covariance matrix, the QF distance is called the Mahalanobis distance [22]. If the bin-similarity matrix is positive-semidefinitive (PSD), A matrix can be expressed as $A = LL^T$ for some real matrix L. Thus, the distance can be computed as the Euclidean norm between linearly transformed vectors:

$$QF^A(P, Q) = \|LP - LQ\|_2 \tag{1.8}$$

In this case the QF distance is a **psuedo-metric**

1.5 Related Work

Our method builds in a novel direction on the success of previous metric learning approaches. As Weinberger and Saul [24] conjectured, more adaptive transformations of the input space can lead to improved performance. Our method allows to enlarge the number of the learned parameters, while the computation of the distance between two never-seen examples is only linear in the dimension.

Chopra et al. [2] proposed to learn a convolutional neural net as a nonlinear transformation before applying the 2 norm. They showed excellent results on image data. Babenko et al. [] suggested a boosting framework for learning non-Mahalanobis metrics. They also presented excellent results on image data. These methods are non-convex and thus they might suffer from local minimas and training is sensitive to parameters.

Kernel methods were also proposed in order to learn a Mahalanobis distance over nonlinear transformations of the data [1, 7, 9]. Computing such a distance between two vectors scales linear in the number of training examples, which makes it impractical for large datasets. Computing our ID distances does not depend on the number of training examples.

A family of non-Mahalanobis distances recently proposed is the Quadratic-Chi (QC) [15]. The QC family generalizes both the Mahalanobis distance and the 2 distance. A QC distance have parameters that can be learned. However, a serious limitation is that it can only model 2 -like distances. In addition, it is applicable only to non-negative vectors. Finally, it is non-convex with respect to its parameters, so learning them is hard.

Rosales and Fung [5] also propose learning metrics via linear programming. However, while we learn a non-Mahalanobis distance, their method learns a subfamily of Mahalanobis distance. That is, their method is restricted to learning a Mahalanobis distance which is parameterized with a diagonal dominant matrix.

Danfeng et al. [3] displays a Quantized Kernels metrics learning methods concludes additive and block-wise kernels learning. Our method refers to any multi-dimensional distance learning problem, not only blocks of objects (such as SIFT descriptor maps around any interest point of an image)

1.6 Contribution

In this novel work, we present an efficient method for embedding either single object or pairs of objects. This method applies a semi-metric learning for a given data space. This method is a generalization of the single-dimensional Interpolated-Discretized (ID) distance, presented by Dr. Ofir Pele [14]. In this work we embed pairs of objects jointly, which for our best of knowledge is the debut embedding method for such purpose. In this work a novel attitude for upgrading IDD embedding procedure to n-dimensional IDD, while maintaining its basic semi-metric, non-euclidean properties, and also contributes the ability of applying “physical” constraints during the embedding process to maintain our method continuous and linearly computed.

Chapter 2

ID Embedding

We now describe the general embedding process for both single objects and object pairs. This is the core process applied on several tasks in our work, such classification, regression, pairs matching etc. Our embedding method is assembled from three main phases:

- Discretization
- Interpolation
- Assigning

let us describe each part in the ID sequence:

2.1 Discretization

Discretization phase is performed in order to downgrade complexity of a given machine/metric learning problem. Let us assume we have a very high ordered vectors to classify, for example a 1G ordered vectors dataset $\overrightarrow{v} \in \Re^{10^9}$ would cause struggled learning process due to high memory resources required.

For that reason we downscale problems' dimensions by discretizing the dataset in the following **dimension-wise** method: Each dimension in dataset is clustered and sorted into C_i - dimensional $\overrightarrow{v} \in \Re^{C_i}$ vector.
Any common clustering method may benefit in this, with one exception: The extrema points of the sorted discretized vector must surround the extrema values of the dataset.

2.2 Interpolation

As described above, our IDD function should delivers continuous output for any given valid object/pair of objects. For this purpose we perform interpolation of the given data sample features, where each element among data sample is interpolated by its closest boundaries in the proper discretization vector space. By the following algorithm:

1. For each element find closest bounds among discretization vector
2. Compute coefficients - this will be described further for every scenario, where this phase is actually performs a multidimensional interpolation

2.3 Assigning

This phase assigns the coefficients computed in the last phase, in their proper locations among the embedded (sparse) vector.

Algorithm 1 Embedding Method - General

```
Input: $L$ sized, vectorized n-dimensional dataset
Input: number of centers per dimension - $C$
Output: $\overrightarrow{\phi}$: $L$ sized set, embedded, sparse vectors
  Find centers vectors
  V shall be a set of centers vectors
  for all dim in $n$ do
    $V_{dim} \leftarrow centers \quad vector \quad per \quad dim$
  end for
  Find embedded coefficients for all dataset
  $\overrightarrow{\phi} = C^n$ length empty $\overrightarrow{\phi}$ embedded vectors
  for all $vec$ in $L$ do
    find $vec$ bounding hypercube
    find $vec$ bounding simplex (permutation method)
    $\overrightarrow{\lambda} \leftarrow$ find $vec$ barycentric coefficients
    $\overrightarrow{\hat{\lambda}} \leftarrow$ normalize($\overrightarrow{\lambda}$)
  end for
  Assign
  for all $\overrightarrow{evec}$ in $emb - set$ do
    $inds \leftarrow$ find vertices from hypercube and simplex locations
    for all $i$ in $inds$ do
      $\overrightarrow{evec}(i) \leftarrow \overrightarrow{\hat{\lambda}}(j(i))$ – j is the assigning function between the coef. vector and embedding vector
    end for
  end for
  return $\overrightarrow{\phi}$
```

In the following sections we describe specifically each nuance of each sub-domain of the method. Please notice that the most detailed sub-method in this work is the multidimensional IDD pairs embedding, since it is the most innovative section in this work in our opinion.

Chapter 3

Interpolated Discretized object pairs embedding

We now describe the Interpolated Discretized Distances (IDD) embedding method. As mentioned above, this method uniqueness is applying an embedding method that treats each pair as a **joint** object in its problem. This method may fit any two-objects task such similarity/matching problems etc. Let us initiate by presenting the original single dimensional (IDD-1D) work [14], then the expansion of this work into multidimensional (IDD-ND) scenario - general distance embedding is described.

3.1 1D case

Our method's objective is to find an embedding function such:

$$ID : \Re^n \times \Re^m \rightarrow \Re^d \tag{3.1}$$

, which applies the following distance function by multiplying with a learned weights vector $\overrightarrow{w}$ (learned vector)

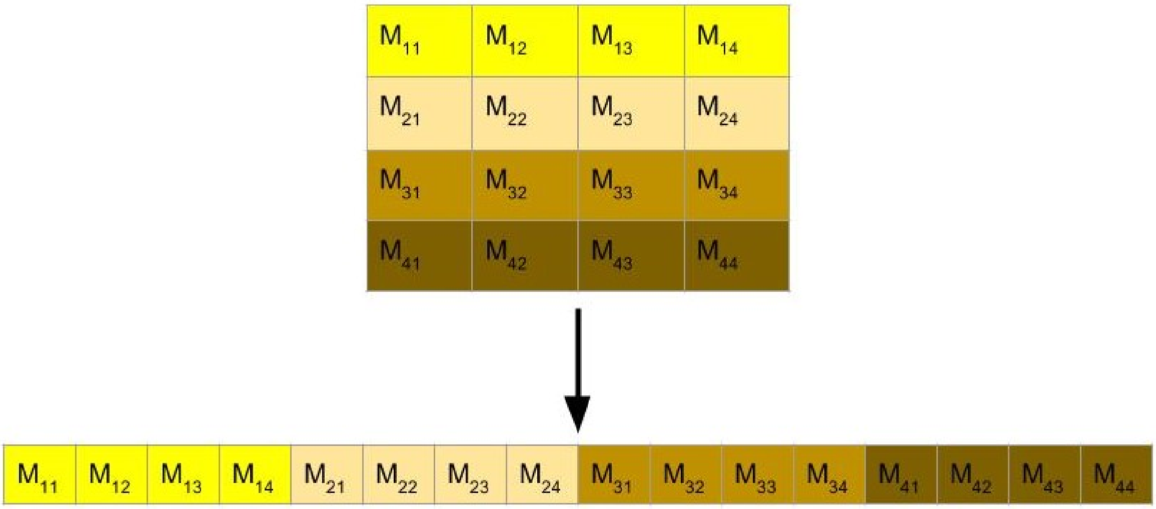
$$d(\overrightarrow{x_1}, \overrightarrow{x_2}) = ID(\overrightarrow{x_1}, \overrightarrow{x_2}) \times \overrightarrow{w} \tag{3.2}$$

this embedding shall obtain semimetric constraints applied.

3.1.1 Discretization

$$\text{W} = \begin{array}{cc} & \left(c_{1(1)}, \dots, \dots, \dots, c_{1(n)}\right) \\ \begin{pmatrix} c_{2(1)} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_{2(m)} \end{pmatrix} & \begin{pmatrix} s_{1,1} & \cdots & \cdots & \cdots & s_{1,m} \\ \vdots & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ \vdots & & & & \vdots \\ s_{n,1} & \cdots & \cdots & \cdots & s_{n,m} \end{pmatrix} \end{array}$$

as described in 2, $1D-IDD$ describes bin-to-bin distance between two samples from single dimensional spaces. Each dimension is clustered and sorted into C_i - dimensional: $\overrightarrow{c} \in \Re^{C_i}$ vector. The vector's pair defines the shape of a distance matrix $W \in \Re^{C_i \times C_j}$, where each element W_i, j describes the distance between two cluster centers form vectors - v_a, v_b



M_{11}	M_{12}	M_{13}	M_{14}
M_{21}	M_{22}	M_{23}	M_{24}
M_{31}	M_{32}	M_{33}	M_{34}
M_{41}	M_{42}	M_{43}	M_{44}

M_{11}	M_{12}	M_{13}	M_{14}	M_{21}	M_{22}	M_{23}	M_{24}	M_{31}	M_{32}	M_{33}	M_{34}	M_{41}	M_{42}	M_{43}	M_{44}

FIGURE 3.1: Close-up of a gull

3.1.2 Interpolation

As described above , our IDD function provides continuous output for any given valid object/pair of objects. For this purpose we perform interpolation of the given data sample features within the W matrix space for each data sample by the following process:

- extract the closest vertices from W matrix to the feature sample.
- calculate four (2 per feature) coefficients per sample, each represented by the normalized surface of the opposite triangle to a vertex (1D coefficients calculation will be described in section).
- 1D-IDD is computed by applying inner product between the sparse coefficient vector and their corresponding vertices vector:

$$1DIDD = \sum_{t=1}^{3} \alpha_{a(t),b(t)} \times W_{a(t),b(t)} \tag{3.3}$$

where:
α is the coefficients sparse vector , representing the interpolation result per feature vector
a, b are parametrization function for both α, W t is the scanning index for all arguments among this expression

Figure 3.1 shows a photograph of a gull. Please notice that there are only 4 elements different than zero at α, so this expression represents the non-zero elements only provides value to $1DIDD$ expression

a, b parameterizations are described as:

- $$a_{(1)} = a(2) = argmax_{(c_i)}\{v_{c_i} \leq x_i\} \tag{3.4}$$
- $$a_{(3)} = a(4) = argmin_{(c_i)}\{v_{c_i} \geq x_i\} \tag{3.5}$$

- $$b_{(1)} = b_{(3)} = argmax_{(c_j)}\{v_{c_j} \leq x_j\} \tag{3.6}$$

- $$b_{(2)} = b_{(4)} = argmin_{(c_j)}\{v_{c_j} \geq x_j\} \tag{3.7}$$

3.1.2.1 Interpolation Coefficients extraction - 1D

Let us describe how does coefficients vectors are calculated from a data sample (assembled from a pair of samples) and a discretization matrix.

For a given data pair, bounding square (2D-cube) is assembles from the clusters vectors per feature. this square is divided into 2 triangles (simplices) along its main diagonal, as described in figure 3.2.

The containing triangle of the data sample is divided to 3 sub-triangles by applying direct lines between the data sample and each vertex of the relevant triangle. Each sub-triangle relative surface represents the coefficient of the opposite vertex 3.2.
In the 1D case, there would be just **3** non-zero elements in the ID coef. vector, since there are 3 affecting sub-triangles (the forth vertex of the cell is zeroed like any other vertex in the output ID vector) .

Single dimensional formation IDD applies semi-metrics properties as described back in 1

3.1.3 Assigning

Now that we have ID coefficients (sparse) vector, we may assign it to the ID output vector shape. ID output vector is sized by the flatten vector of the discretization matrix W, which may be flatten row-wise/column-wise 3.3.
Each vertex in the output vector receives the value calculated for its equivalent index at the opposite triangles surfaces phase.

3.2 multidimensional case

We now address describing the generalization of the single dimension IDD method to n-dimensional object pairs embedding.
For this we should adapt a different embedding attitude, since it should obtain multi-dimensional embedding, unlike the triangles relative surfaces process performed in the 1d scenario.
The selected coefficients calculation process for our embedding is the Barycentric (center of mass) Coordinates [12] of a given vector in a 2n dimensional space (2n since we embed pairs of objects for distance/similarity calculation).

3.2.1 Definition

The general expression of $NDIDD$ would appear to be:

$$NDIDD = \sum_{t=1}^{2n!} \alpha_{a(t),b(t)} \times W_{a(t),b(t)} \tag{3.8}$$

In this scenario, a, b are the multi-dimensional parametrization functions, which correlates between the coefficients vector and the learned weights vector.

3.2.2 Discretization

Given a n-dimensional vectors dataset, we first discrete the data range/space of each dimension, into C discretization values. C may vary among dimensions. This step is equivalent to the 1D scenario. At the 1D scenario, a 2D matrix was generated, which represents the distance between a pair of elements.

at this stage, we may use several methods for discovering optimal number of centers per dimension:

- cross validation on k-folded data [8]. a decribed algorithm is described below 2.
- manual selection - dimension independent

Algorithm 2 cross-validation (cv) algorithm for single dimension centers extraction

```
Input: $\overrightarrow{v}$: single dimension from a given dataset
Input: $k$: number of folds for cv operation
Input: $\overrightarrow{c_{range}}$: range of output centers to look in
Output: $n$: number of optimal centers
  $groups \leftarrow$ split $\overrightarrow{v}$ into $k$ even segments
  for all $c$ in $\overrightarrow{c_{range}}$ do
    $c_{max} \leftarrow 0$
    for all $i$ in $k$ do
      $testgroup \leftarrow groups[i]$
      $traingroup \leftarrow groups[! = i]$
      $clusters \leftarrow$ k-means($traingroup$)
      $score \leftarrow$ silhouette($testgroup$, $clusters$)
      $c_{max}$ += $score$
    end for
    if $score$<$c_{max}$ then
      $score_{max} \leftarrow c_{max}$
      $C \leftarrow c$
    end if
  end for
  return $C$
```

Now we address the n-dimensional scenario, by obtaining a $2n$ dimensional tensor, representing the distance between a pair of n dimensional vectors.

3.2.3 Interpolation

Next phase is interpolating every dataset sample, or any tested sample, in order to embed it using our method.

let us assign a pair of n-dimensional vectors $x_1, x_2 \in \Re^n$.
$NDIDD$ embedding handles this pair as a **joint 2n dimensional vector**, flatten in the following order:

$$\overrightarrow{p} = [x_1^1, x_2^1, ..., ..., x_1^n, x_2^n], \qquad \overrightarrow{p} \in \Re^{2n} \tag{3.9}$$

for the further process description we will treat $\overrightarrow{p}$ as our data object

3.2.3.1 find bounding hypercube

First, we place the sampled vector $\overrightarrow{p}$ within its bounding 2n-hypercube, same as performed in the single dimensional scenario 3.1.2.

3.2.3.2 find bounding simplex

Next stage is discover the simplex (equivalent to triangle in 1d scenario) containing point $\overrightarrow{p}$. A 2n-dimensional hypercube is assembled from $(2n)!$ vertices. Assuming the vertices values are normalized to a "unit hypercube" - containing only $0/1$ values in elements, any vertex applies a permutation as follows:

$$0 \leq p_{t(1)} \leq p_{t(1)} \leq \cdots \leq p_{t(2n-1)} \leq p_{t(2n)} \leq 1 \tag{3.10}$$

[19]

where:

$t_{(i)}$ is a permutation function.

so how we select the right simplex (and permutation) for a given point p? Each point $\overrightarrow{p}$ obeys a unique permutation. A certain permutation defines the right simplex vertices. For each set of correct vertices that obeys a certain permutation, the extreme vertices, all zeros/all ones values, always included in the right vertices account.

3.2.3.3 calculate sub-volumes of all simplices involved

Next phase is calculating the relative volumes (equivalent to 1 dimensional relative surfaces) of all sub simplices.

Given a simplex assembled from $T = 2n + 1$ set of vertices, which bounds a point $\overrightarrow{p}$, we calculate the volumes of T sub-simplices, so every simplex is assembled from $T - 1$ vertices plus $\overrightarrow{p}$.

These normalized volumes represents the coefficients of the missing (or counter in the T space) vertex.

3.2.3.4 Simplex Volume Calculation

Given $T = 2n + 1$ simplices of $N = 2n$ dimension, a general expression for the volume contained between its vertices would be:

$$V_{[r^1,r^2,r^3,\ldots,\ldots,r^T]} = \frac{1}{(2n)!} \times \begin{vmatrix} 1 & r_1^1 & r_2^1 & \ldots & r_{2n}^1 \\ 1 & r_1^2 & \ddots & \ddots & r_{2n}^2 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & r_1^T & \ldots & \ldots & r_{2n}^T \end{vmatrix} \tag{3.11}$$

We calculate the volume for every vertex in the simplex as follows: Given a point $\overrightarrow{p}$ and a set of vertices $r^1, r^2, r^3, ..., r^T$, the volume correspond to every vertex r^i is:

$$\lambda_i = V_{[r^1,r^2,p,\ldots,\ldots,r^T]} = \frac{1}{(2n)!} \times \begin{vmatrix} 1 & r_1^1 & r_2^1 & \ldots & r_{2n}^1 \\ 1 & r_1^2 & \ddots & \ddots & r_{2n}^2 \\ \vdots & p_1 & p_2 & \ddots & p_{2n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & r_1^T & \ldots & \ldots & r_{2n}^T \end{vmatrix} \tag{3.12}$$

3.2.3.5 Efficient method - calculating Barycentric Coordinates (BCC)

A more efficient way (time complexity $O(n)$) providing the coefficients to a given point is by using barycentric coordinates of a point $\overrightarrow{p}$. Given a point $\overrightarrow{p} \in \Re^{2n}$, the following formulation applies under the assumption the hypercube bounding the point is mapped to a unit hypercube, the bounding simplex of the point may be:

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}$$

so the equation we shall use is this:

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \lambda_0 + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \lambda_1 + \cdots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \lambda_{2n-1} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \lambda_{2n} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \quad , \quad \sum_{i=0}^{2n} \lambda_i = 1 \tag{3.13}$$

notice that number of both vertices and barycentric coordinates are $T = 2n + 1$.

3.2.3.6 O(n) solution

We solve this equation by using the gradation of the vertices along the left side of the equation.

Let $i = 2n, \ldots, 2, 1$. The recursive formulation for the solution of the equation would be the follows:

$$\lambda_i = p_{2n-i+1} - \sum_{j=i+1}^{2n} \lambda_j \tag{3.14}$$

and the remained coefficient extracted from the final step:

$$\lambda_0 = 1 - \sum_{j=1}^{2n} \lambda_j \tag{3.15}$$

3.2.3.7 Methods identity for n=2

Let us demonstrate the identity of BCC calculation methods for $n = 2$ dimensions:

3.2.3.8 Volumes method

let our sample be $\overrightarrow{p} = [p_1, p_2, p_3, p_4]$, and let us assume permutation:
$p_1 \leq p_2 \leq p_3 \leq p_4$.
the vertices obeys to the given permutation are:

- $r_0 = [0, 0, 0, 0]$
- $r_1 = [0, 0, 0, 1]$
- $r_2 = [0, 0, 1, 1]$
- $r_3 = [0, 1, 1, 1]$
- $r_4 = [1, 1, 1, 1]$

We begin with λ_4 calculation, since it is straightforward from matrix's gradation

$$\lambda_4 = V_{[r^0,r^1,r^2,r^3,p]} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & p_1 & p_2 & p_3 & p_4 \end{vmatrix} = p_1 \tag{3.16}$$

for λ_3 we use the proceed the algorithm:

$$\lambda_3 = V_{[r^0,r^1,r^2,p,r^4]} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & p_1 & p_2 & p_3 & p_4 \\ 1 & 0 & 1 & 1 & 1 \end{vmatrix} = p_2 - p_1 = p_2 - \lambda_4 \tag{3.17}$$

λ_2:

$$\lambda_2 = V_{[r^0,r^1,p,r^3,r^4]} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & p_1 & p_2 & p_3 & p_4 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{vmatrix} = p_3 - p_2 = p_3 - (\lambda_3 + \lambda_4) \tag{3.18}$$

λ_1:

$$\lambda_1 = V_{[r^0,p,r^2,r^3,r^4]} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & p_1 & p_2 & p_3 & p_4 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{vmatrix}$$

$$= p_1 \cdot \begin{vmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} - p_2 \cdot \begin{vmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} + p_3 \cdot \begin{vmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} - p_4 \cdot \begin{vmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

$$= p_4 - p_3 = p_4 - (\lambda_2 + \lambda_3 + \lambda_4)$$

(3.19)

vectors dimension[n]	vertices [(2n)!]	simplex vertices [2n+1]	simplices
1	4 (exceptional private case)	3	2
2	24	5	4
3	720	7	6
4	40320	9	8
n	(2n)!	2n+1	2n

TABLE 3.1: higher dimensions' number of simplices and vertices

for λ_0 we apply the final step expression:

$$\lambda_0 = V_{[p,r^1,r^2,r^3,r^4]} = \begin{vmatrix} 1 & p_1 & p_2 & p_3 & p_4 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{vmatrix}$$

$$= 1 - p_1 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} + p_2 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} - p_3 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} + p_4 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix}$$

$$= 1 - p_4 - p_3 = 1 - (\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4)$$

(3.20)

Final coefficients vector produced by volumes method would be:

$$\begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} = \begin{bmatrix} 1 - (\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4) \\ p_4 - (\lambda_2 + \lambda_3 + \lambda_4) \\ p_3 - (\lambda_3 + \lambda_4) \\ p_2 - \lambda_4 \\ p_1 \end{bmatrix} \tag{3.21}$$

3.2.3.9 Recursive method

based on the formulation developed above 3.14, we extract the now extract the λ coefficients:
$\lambda_4 = p_1$
$\lambda_3 = p_2 - \lambda_4$
$\lambda_2 = p_3 - \lambda_3 - \lambda_4$
$\lambda_1 = p_4 - \lambda_2 - \lambda_3 - \lambda_4$

and for λ_0 simply apply the final step which provides **identical** result to the volume method:
$\lambda_0 = 1 - \lambda_1 - \lambda_2 - \lambda_3 - \lambda_4$

3.2.4 Assigning

After calculating coefficients vector for $2n + 1$ related vertices (from bounding simplex) of a data sample p, we now address embedding those coefficients into a single, flatten, sparse vector $\alpha \in c^{2n}$, where c is the number of means per dimension, and n is the dimension of a single object from an object pair.

The only non-zero values in the embedded vector would be the ones related to the bounding simplex of the sample $\overrightarrow{p}$.

–insert figure Fig - embedding from matrix to sparse vector

The embedding process could be easily generalized if c is vectorized to values - set, then the dimension of α would be:
$\prod_{i=1}^{2n} c_i$, which equivalent to the number of vertices in the "discretized" space of the problem. In order to finally receive a distance/similarity/dis-similarity function, we apply inner product between α vector and a learned weights vector (equivalent to the W matrix from the $1D$ scenario).

Algorithm 3 Embedding Method for ID N-Dimensional Pairs dataset

```
Input: $L$ sized, vectorized n-dimensional pairs dataset
Input: set of centers per dimension - $C$
Output: $\overrightarrow{\phi}$: $L$ sized set, embedded, sparse vectors
  Find centers vectors
  V shall be a set of centers - vectors
  for all dim in $n$ do
    $V_{dim} \leftarrow centers \quad vector \quad per \quad dim$
  end for
  Find embedded coefficients for all dataset
  $\overrightarrow{\phi} = C^{2n}$ length empty $\overrightarrow{\phi}$ embedded vectors
  concat all pairs into a single vector $\overrightarrow{p}$
  for all $\overrightarrow{p}$ in $L$ do
    find $\overrightarrow{p}$ bounding hypercube
    find $\overrightarrow{p}$ bounding simplex (permutation method)
    $\overrightarrow{\lambda} \leftarrow$ find $\overrightarrow{p}$ barycentric coefficients
    $\overrightarrow{\hat{\lambda}} \leftarrow$ normalize($\overrightarrow{\lambda}$)
  end for
  Assign
  for all $\overrightarrow{\phi}$ in $emb - set$ do
    $inds \leftarrow$ find vertices from hypercube and simplex locations
    for all $i$ in $inds$ do
      $\overrightarrow{\phi}_{(i)} \leftarrow \overrightarrow{\hat{\lambda}}(j(i))$ – j is the assigning function between the coef. vector and embedding vector
    end for
  end for
  return $\overrightarrow{\phi}$
```

3.3 Non-euclidean non-embeddable Metrics example

Let us display an example of a useful application to IDDND method. As described in the introduction chapter, IDDND method overcomes two main issues:

1. **distortion** issues cause when applying euclidean metrics on a given arbitrary dataset.
2. **Metrics** are not necessarily applied on dissimilarities problems

We demonstrate a theoretical use case constrained by those 2 conditions, and see how our method would overcome those and fits a proper model to a non-embeddable dataset.

This theoretical set form applies a non-euclidean embeddable metrics, which we now prove that a euclidean metrics is unable to embed it.

Based on euclidean metrics assumptions, assumed there is an integer k , such that a function f applies:

$f \ : \ \{\overrightarrow{x_1}, \overrightarrow{x_2}, \overrightarrow{x_3}, \overrightarrow{x_4}\} \rightarrow \Re^k$

Where $\overrightarrow{x_i}$ are samples from each one of data centers, and f preserves the distances.

As triangle-inequality is tight for: $\overrightarrow{x_4}, \overrightarrow{x_1}, \overrightarrow{x_2}$,

and $f(\overrightarrow{x_4}), f(\overrightarrow{x_1}), f(\overrightarrow{x_2})$ are collinear in $\Re^k$

From set formation symmetry property, the same co-linearity applies on:
$f(\overrightarrow{x_3}), f(\overrightarrow{x_1}), f(\overrightarrow{x_2})$

This common co-linearity of those tuples leads to the following equation:
$\|f(\overrightarrow{x_4}) - f(\overrightarrow{x_3})\|_2 == 0$

But this fact is contradicting that
$d(\overrightarrow{x_4} \ , \overrightarrow{x_3}) = 2$

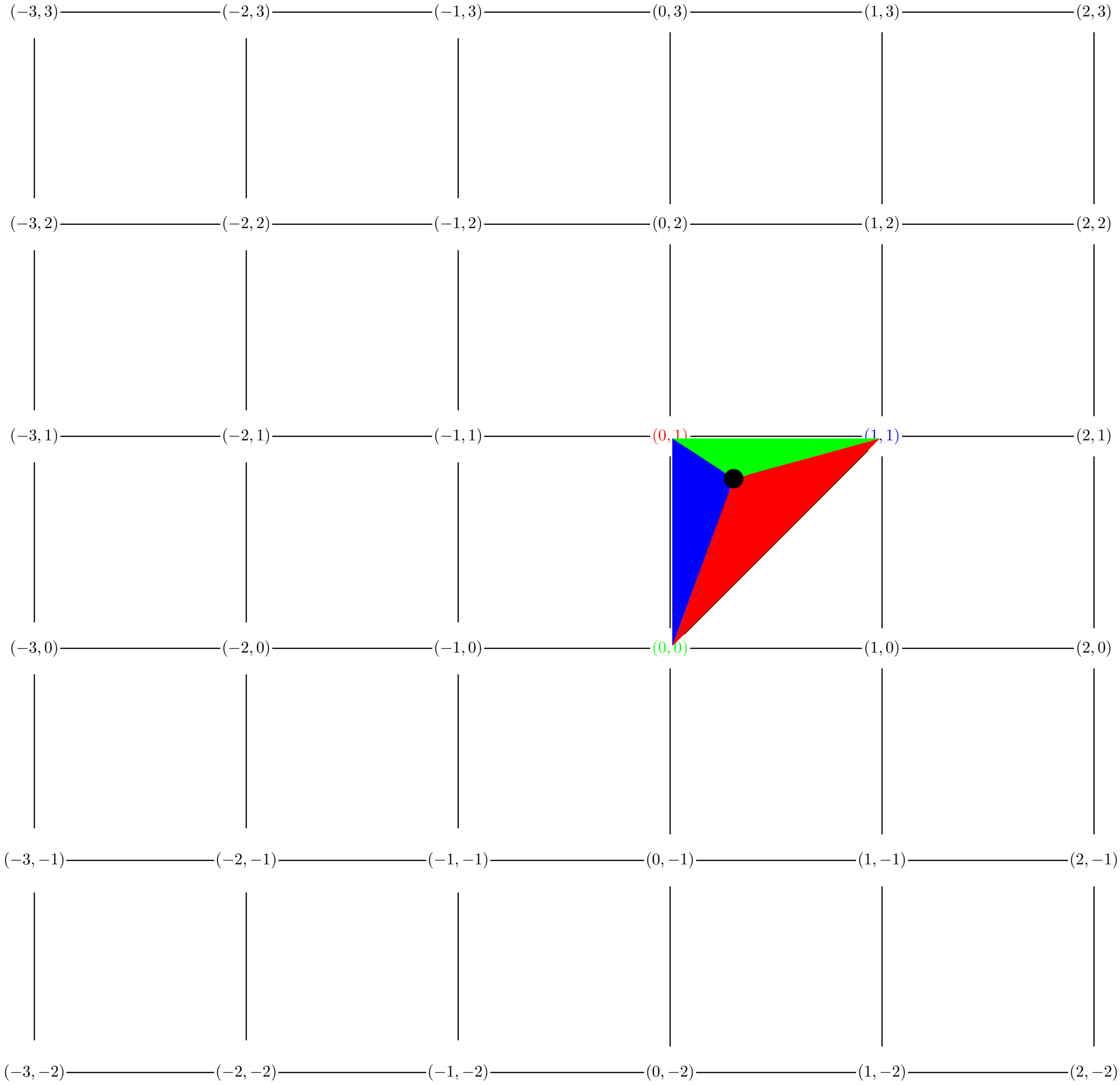


FIGURE 3.2: this matrix scheme describes a given data sample coefficients calculation according to its containing hypercube and simplex.The examined data sample is $(0.3, 0.8)$, so its containing hypercube would be the cell between $[0, 1]$. In this hypercube we search the containing simplex (triangle in 1D case), which occurs to be the upper triangle among the 2 main diagonal triangles. By assembling 3 sub-triangles using the data sample and the vertices we may extract the proper coefficients for the ID vector. Each normalized sub-triangle surface represents the contrary vertex's coefficients. For example the blue triangle surface correlates with the $(1, 1)$ vertex's coefficient , the green surface correlates with the $(0, 0)$ vertex's coefficient and the red surface correlates with the $(0, 1)$ vertex's coefficient.

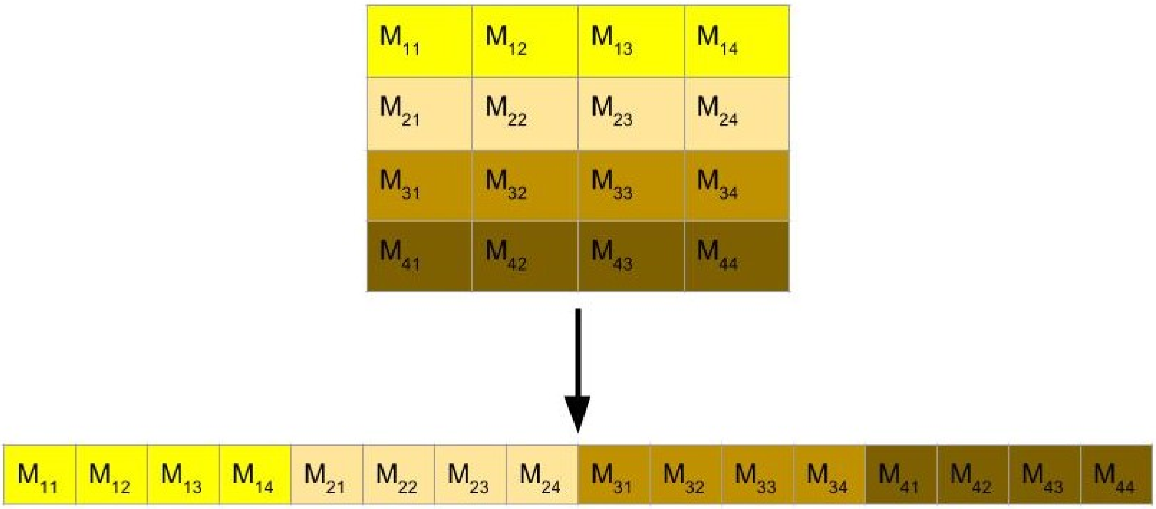

M_{11} M_{12} M_{13} M_{14}
M_{21} M_{22} M_{23} M_{24}
M_{31} M_{32} M_{33} M_{34}
M_{41} M_{42} M_{43} M_{44}
M_{11} M_{12} M_{13} M_{14} M_{21} M_{22} M_{23} M_{24} M_{31} M_{32} M_{33} M_{34} M_{41} M_{42} M_{43} M_{44}

FIGURE 3.3: flattening 2d matrix to row-wise 1d vector

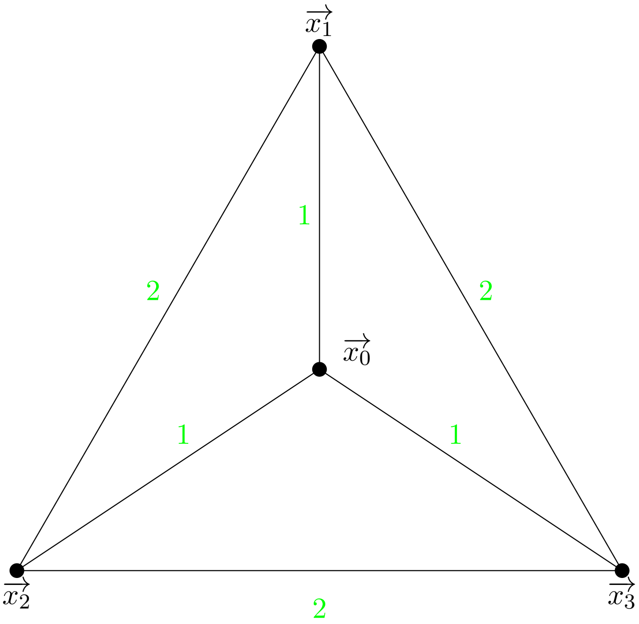


FIGURE 3.4: non-embeddable dataset scheme, describing a 2D triangled set, where 4 centers represents 4 data groups which their average distances shown in green

Chapter 4

Interpolated Discretized Single objects Embedding

Data objects may describe an abstracted format of any data type exists, such images, video, audio, text etc. **Single** objects embedding may assist in classification and regression tasks, by emphasizing differences among data samples, without spending lots of time and memory.
In this section we describe how we perform our ID method on such single vectors domain.

4.1 Discretization

We begin by dataset discretization, which is equivalent in all the use cases displayed in our method. Dataset is being discretized by performing clustering on its dimensions and find C centers for discretization. This step should maintain the dataset extreme values within the extreme values of the C vector (so in the interpolation phase there will be valid values for all data elements 4.1.

4.2 Interpolation

Interpolation of a given dataset, after extracting its discretization centers, is performed by the following sequence:

- Find bounding hypercube
- Find bounding simplex
- Find sample's correlated coefficients per simplex vertices as described at 3.1.2
- Convert coef. vector to normalized format. $v_i \in [0, 1]$ by dividing with the volume of the simplex

$$\begin{array}{c} \big(v_1, v_2, \ldots, \ldots, v_{n-1}, v_n\big) \\ \begin{pmatrix} m_{1,1} & \cdots & \cdots & \cdots & m_{1,n} \\ \vdots & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ m_{C-1,1} & & & & m_{C-1,n} \\ m_{C,1} & \cdots & \cdots & \cdots & m_{C,n} \end{pmatrix} \end{array}$$

FIGURE 4.1: discretization points set W as built from a single dimensional dataset. this set is vectors' length may vary among dimensions

4.3 Assigning

Here we proceed embedding process by assigning the normalized coefficients vector to their indices in the embedded vector. In the 1d scenario, the embedded vector would be sized as centers number per dimension - C, powered by dimension length - n

Algorithm 4 Embedding Method for ID N-Dimensional single vectors dataset

```
Input: $L$ sized, vectorized n-dimensional dataset
Input: set of centers per dimension - $C$
Output: $\overrightarrow{\phi}$: $L$ sized set, embedded, sparse vectors
  Find centers vectors
  V shall be a set of centers - vectors
  for all dim in $n$ do
    $V_{dim} \leftarrow centers \quad vector \quad per \quad dim$
  end for
  Find embedded coefficients for all dataset
  $\overrightarrow{\phi} = C^n$ length empty $\overrightarrow{\phi}$ embedded vectors
  for all $\overrightarrow{p}$ in $L$ do
    find $\overrightarrow{p}$ bounding hypercube
    find $\overrightarrow{p}$ bounding simplex (permutation method)
    $\overrightarrow{\lambda} \leftarrow$ find $\overrightarrow{p}$ barycentric coefficients
    $\overrightarrow{\hat{\lambda}} \leftarrow$ normalize($\overrightarrow{\lambda}$)
  end for
  Assign
  for all $\overrightarrow{\phi}$ in $emb - set$ do
    $inds \leftarrow$ find vertices from hypercube and simplex locations
    for all $i$ in $inds$ do
      $\overrightarrow{\phi}_{(i)} \leftarrow \overrightarrow{\hat{\lambda}}(j(i))$ – j is the assigning function between the coef. vector and embedding vector
    end for
  end for
  return $\overrightarrow{\phi}$
```

Chapter 5

Learning

In this section described the learning phase of the ID method.
Our Learning section refers to the pairs embedding use case. Learning a single vectors dataset will be described further, since it is the simpler scenario and quite similar.
In general, since our method is embedding objects or object pairs to sparse vectors set, we can use this quality in order to accelerate learning phase of the process.

5.1 Learning Binary Classification Function

As described above 2, our current method is handling similarity detection between two n-dimensional vectors. This can of course be generalized to any classification/clustering matter.
Let X be a set of raw data vectors. Each vector in this set may represent a single object, for any type of classification analysis.
In this scenario we obtain object pairs **similarity** problem.

Let us assign an indexing system for this labeled pairs dataset as follows:

$$P = \begin{bmatrix} p_{11} & p_{12} \\ \vdots & \vdots \\ p_{i1} & p_{i2} \\ \vdots & \vdots \\ p_{k1} & p_{k2} \end{bmatrix} \overrightarrow{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_k \end{bmatrix} \tag{5.1}$$

Where P set refers to X set indices and $\overrightarrow{y}$ refers to the vectors label as follows:

$$y_i = \begin{cases} -1 \;\; if\; \overrightarrow{x_{pi1}}\; and\; 1, \overrightarrow{x_{pi2}}\; are\; similar \\ +1 \;\; if\; 2; S) = [ID(\overrightarrow{x_{pi1}}\; and\; 1_{[}S_1], \overrightarrow{x_{pi2}}\; are\; non - similar \end{cases} \tag{5.2}$$

We define a classification (similarity) function:

$$similar(\overrightarrow{x_1},_{[}\, S_g], \overrightarrow{x_2}) = \begin{cases} -1 \;\; if\; d(\overrightarrow{x_1}, \overrightarrow{x_2}) \;=\; IDD(\overrightarrow{x_1}, \overrightarrow{x_2}) \cdot \overrightarrow{w} < t \\ +1 \;\; otherwise \end{cases} \tag{5.3}$$

Where a pair of vectors is similar if and only if their ID distance result is smaller than a given threshold parameter (t). This function is identical to a classification method of a standard binary SVM classification method [21], which looks like the following:

Where:
t - threshold which learned by an optimization process - weight vector of the problem which is also learned by an optimization process.

We now address describing their optimization of the learning step for achieving optimal model for a certain data set.

5.1.1 Efficient Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) [9] is a stochastic approximation of the gradient descent optimization method for minimizing an objective function that is written as a sum of differentiable functions.

SGD is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as our learning function.

As Shalev-Shwartz et al. POLA [17], we can learn our weights (including t parameter):

$$\overrightarrow{w}^{opt}, t^{opt} = \underset{\overrightarrow{w},t}{argmin}(\frac{1}{2}\left\|\overrightarrow{w} - \overrightarrow{w}^{reg}\right\|_2^2 + C\sum_{i=1}^{k} max(1 - (IDD(\overrightarrow{x_{p_i}1}, \overrightarrow{x_{p_i}2}) \cdot \overrightarrow{w} - t)y_i, 0)) \tag{5.4}$$

Where:

- $\overrightarrow{w}^{reg}$ represents a regularizer distance for the vertices, e.g. , L^1
- C represents a decay factor applied for convergence control in the optimization process.

Let us defined an optimized implementation adapted to the similarity problem.

Naive implementation of Stochastic Gradient Descent will result in time complexity of $O(c^{2n})$,
due to regularizer appearance in the equation:

$$\frac{\partial\frac{1}{2}\frac{1}{k} \cdot \left\|\overrightarrow{w} - \overrightarrow{w}^{reg}\right\|_2^2}{\partial\overrightarrow{w}} = \frac{1}{k}(\overrightarrow{w} - \overrightarrow{w}^{reg}) \tag{5.5}$$

We can use an **overcomplete** representation of the weights vector $\overrightarrow{w}$ in order to reduce time complexity of the regularizer to $O(1)$ and total running time of each SGD step to $O(n)$ (similar trick was used by Shwartz et al. Pegasos paper [18]:

$$\overrightarrow{w} = \beta \cdot \hat{\overrightarrow{w}} + \gamma \cdot \overrightarrow{w}^{reg} \tag{5.6}$$

So instead of the common SGD weights update:

$$\overrightarrow{w}^{updated} = \overrightarrow{w} - \frac{\alpha}{tk}(\overrightarrow{w} - \overrightarrow{w}^{reg}) = (1 - \frac{\alpha}{tk})\overrightarrow{w} + \frac{\alpha}{tk}\overrightarrow{w}^{reg} \tag{5.7}$$

With the new representation we can write the updated weights as follow:

$$\overrightarrow{w}^{updated} = (1 - \frac{\alpha}{tk})(\beta \cdot \hat{\overrightarrow{w}} + \gamma \cdot \overrightarrow{w}^{reg}) + \frac{\alpha}{tk}\overrightarrow{w}^{reg} = ((1 - \frac{\alpha}{tk})\beta)\hat{\overrightarrow{w}} + ((1 - \frac{\alpha}{tk})\gamma + \frac{\alpha}{tk})\overrightarrow{w}^{reg} \tag{5.8}$$

Which allows to separate both coefficients of the overcomplete representation:

$$\beta^{updated} = (1 - \frac{\alpha}{tk})\beta \quad \gamma^{updated} = (1 - \frac{\alpha}{tk})\gamma + \frac{\alpha}{tk} \tag{5.9}$$

5.2 Learning Regression Function for similarity/distance problems

As described above 2, our current method handles similarity detection between two n-dimensional vectors.
This theorem can of course be generalized to any classification/clustering matter.

Let X assigned as a matrix of examples:

$$X = \begin{bmatrix} \overrightarrow{x_1} \\ \vdots \\ \overrightarrow{x_u} \end{bmatrix}$$

a matrix of pairs of indices and their labels vector:

$$P = \begin{bmatrix} p_{11} & p_{12} \\ \vdots & \\ p_{i1} & p_{i2} \\ \vdots & \\ p_{k1} & p_{k2} \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_k \end{bmatrix}$$

where:
$y_i \in \mathbb{R}^{+0}$ is the dissimilarity label (continuous).

The optimization function will look as follows:

$$\vec{w}^{\text{opt}} = \underset{\vec{w} \geq 0}{\text{argmin}} \left(\frac{1}{2} ||\vec{w} - \vec{w}^{\text{reg}}||_2^2 + C \sum_{i=1}^{k} \left(\text{ID}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) \cdot \vec{w} - y_i \right)^2 \right) \tag{5.10}$$

where $\vec{w}^{\text{reg}}$ is a regularizer distance for the vertices *e.g.* L^1

We can constrain ID to be

- **symmetric** $\text{ID}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) \cdot \vec{w} = 0$ for $\overrightarrow{x_{p_{i1}}} = \overrightarrow{x_{p_{i2}}}$
- $\text{ID}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) \cdot \vec{w} \neq 0$ for $\overrightarrow{x_{p_{i1}}} \neq \overrightarrow{x_{p_{i2}}}$

replacing $\vec{w} = \vec{w}' + \vec{w}^{\text{reg}}$:

$$\vec{w}^{\text{opt}} = \underset{\vec{w}'}{\text{argmin}} \left(\frac{1}{2} ||\vec{w}'||_2^2 + C \sum_{i=1}^{k} \left(\text{ID}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) \cdot \vec{w}^{\text{reg}} + \text{ID}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) \cdot \vec{w}' - y_i \right)^2 \right) \tag{5.11}$$

Constraints should be in respect to:

$\text{ID}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) \cdot \vec{w}^{\text{reg}} + \text{ID}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) \cdot \vec{w}'$
We can also use directly a regularizer distance d^{reg}:

$$\vec{w}^{\text{opt}} = \underset{\vec{w}'}{\text{argmin}} \left(\frac{1}{2} ||\vec{w}'||_2^2 + C \sum_{i=1}^{k} \left(d^{\text{reg}}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) + \text{ID}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) \cdot \vec{w}' - y_i \right)^2 \right) \tag{5.12}$$

Constraints should be in respect to:

$d^{\text{reg}}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) + \text{ID}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) \cdot \vec{w}'$

Chapter 6

Time Complexity

Time complexity for each step of the $IDDND$ function building is displayed in this section. Learning / training timing was excluded and will be discussed at chapter 8

6.1 Discretization

Discretization implementation is not specified in this paper since we are using a OTS method such $k - means$ clustering algorithm, although This step may be applied by various methods. The common ones are k-means or their variation. Here we refer to $k - means$ [6] based methods. Even though the problem of finding the global optimum of the $k - means$ objective function:

$$arg \min_{s} \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2 \tag{6.1}$$

is NP-hard[].

However, running a fixed number i of iterations of the standard algorithm consumes only:

$O(i\dot{k}\dot{n}\dot{d})$

Where: i - convergence iterations
k - number of means
N - dataset samples
n - object dimension

for n points in d dimensions

6.2 Interpolation

As described above 3, we divide the interpolation phase time complexity for several phases:

6.3 Find the bounding hypercube

Find a given vector′s bounding box is actually find a bounding pair for each element in this vector. Since the means vectors are sorted then this step takes n times (for n dimensions) $log(n)$, which is the time complexity for binary search. In total, time complexity of this step is $O(nlog(n))$

6.4 Find the bounding simplex

Finding the bounding simplex is equivalent to find an obeying permutation over the given vector. Time complexity for this step is therefore: $O(nlog(n))$

. While using common sorting algorithms such Quicksort [7]/Merge Sort [3]/Timsort [16].

6.5 ID coefficients extraction

In the step, we take in matter only the second attitude shown above since it is faster. Computing the coefficients is also possible in $O(n)$. We can compute the first index in $O(n)$ and do additional $O(n)$ updates each with a time complexity of $O(1)$ for computing the other indices, so in total this step take only $O(n)$.

total time complexity for the interpolation step method is:

$$O(nlog(n)) + O(nlog(n)) + O(n) = O(nlog(n))$$

6.6 Assigning

Data assigning of the sparse vector is purely flattening a matrix into a vector shape, which takes $O(1)$ 3.3

Chapter 7

Memory Complexity

In this section we provide a method for applying dimensional reduction of the multidimensional scenario, by grouping significant small subsets of the data and concatenating them after embedding.

7.1 Definition

We define a generalized ID function by:

$$ID_{(}\overrightarrow{x_1}, \overrightarrow{x_2}; S) = [ID(\overrightarrow{x_1}_{[}S_1], \overrightarrow{x_2}_{[}S_1]), \ldots, \ldots, ID(\overrightarrow{x_1}_{[}S_g], \overrightarrow{x_2}_{[}S_g])] \tag{7.1}$$

where S is the defined sub-domain group:

$$\{S_i\}_{i=1}^{g} \quad , \quad S_i \subseteq [1, \ldots, n] \quad , \quad g \in [1, \ldots, n]$$

g represents a sub domain which extracts valid demandable information to the user.

For example when applying ID embedding on **SIFT [11]** descriptors, a certain user may involve only neighbor pairs, where another one may involve just bin-to-bin pairs.

7.1.1 SIFT Dimensions Demonstration

when applying ID on SIFT descriptors (for template matching), $n = 128$, which causes time and memory complexity to be scaled by $n^2 = 128^2 = 16384$. If one user desires, and physically ables to reduce this coefficient, we developed the following generalization.

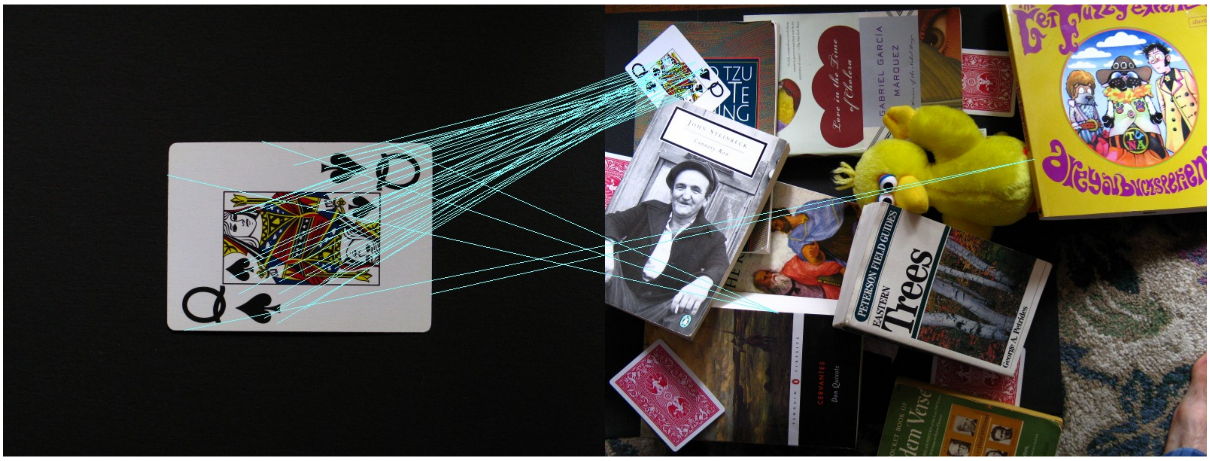


FIGURE 7.1: SIFT algorithm used for template matching. In this example each image is scanned and a set of keypoints (a significant point) is found, and for each keypoints a 3D features vector is calculated. These vectors are being compared between the images, and a similarity is measured among those sets. Here we see that there is a match between both Queen cards in both images, although there are size, orientation, and clearance differences

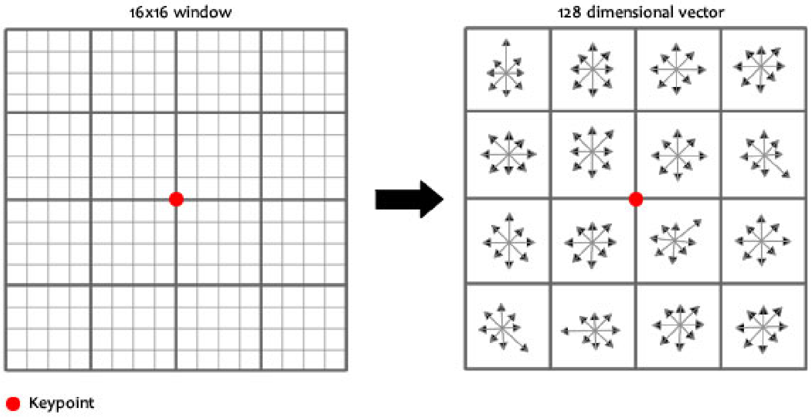


FIGURE 7.2: SIFT keypoint descriptor scheme. A 16x16 neighborhood around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

Chapter 8

Experiments

In this chapter we display a novel usage to our new the ID method attributes and qualities. Here we will use our ID method in order to obtain a **perceptual colors distance metric learning** [15], which is a high demanded function in computer vision region.
Our examination is based and referring to [15] and [26] papers, which developed a local metric embedding method for this problem.

This set of experiments models a perceptual color difference metrics, based on various color samples originated from several cameras, angles, illuminations etc.

Our ID method would be examined in this section solely by the original experiment's accuracy measures - Mean Absolute Error [25], along various parameters such number of centers per dimension, and different datasets assembly.
Further discussion will refer to the various exams may apply on our ID method according to our described theory.

8.1 Experiment Procedure

object pairs ID embedding 3 will be demonstrated by applying its embedding method over pairs which was taken and labeled from the Farnsworth-Munsell 100 hue-test set [13].

8.1.1 Dataset

Dataset of this experiment is assembled from single color patches, which displayed in a set of images, where each image was generated by a specific set of image features, such illumination, camera type (4 different cameras are involved in the dataset creation) , lance type , background etc.

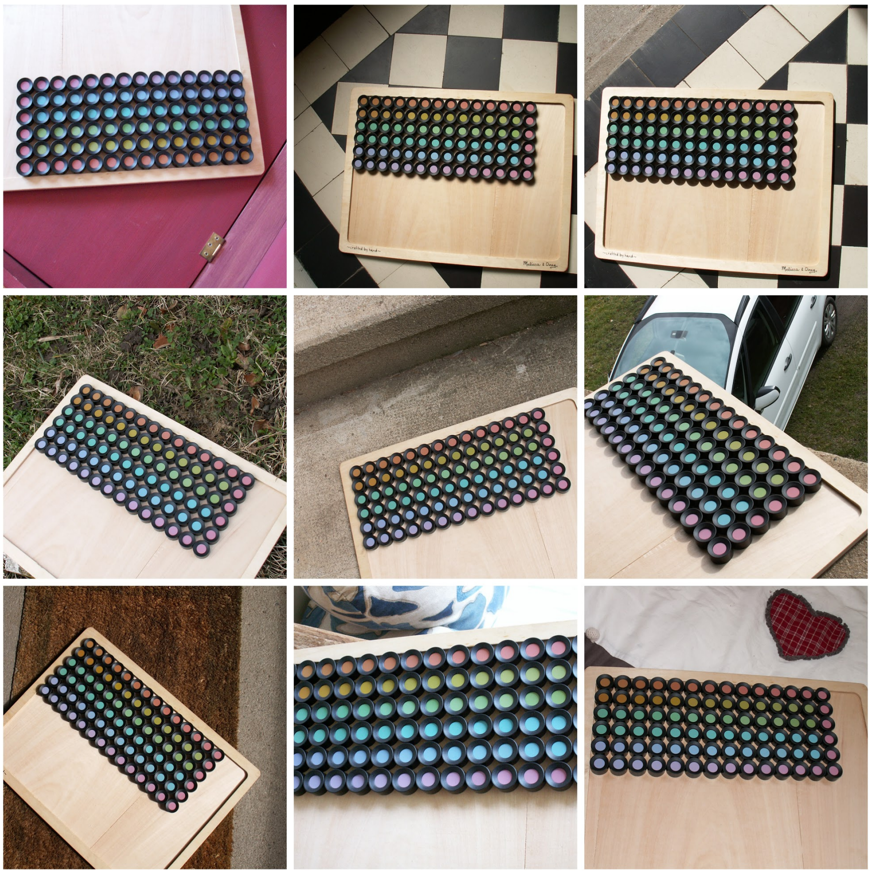


FIGURE 8.1: dataset patches original images formats

For each color patch, a $L^*A^*B^*$ [22] coordinates are given. Since the pairs of patches should be close by their CIELAB [27] coordinates in order to adapt a good color difference assessment, a set in the final dataset is only a set of patches, where its CIELAB euclidean distance is relatively close, i.e. $\Delta E \leq 5$.

8.1.2 Models to train

Then, ID learning method will try to learn some similarity models according to a couple of test conditions:

- **unseen colors** - dividing the entire patches set into train-test sets
- **unseen cameras** - assigning 3 out of 4 cameras sets as train set, and make the remained camera set as test set.
  training set cameras:
  - Kodak DCS Pro 14n
  - Konica Minolta Dimage Z3
  - Nikon Coolpix S6150

  test set camera:
  - Sony DCR-SR32

8.1.3 Interpolation

Data dimension in our scenarios is of course $n = 3$ (for lab/RGB [20] coordinates), and for an object pair the dimension is $2n = 6$.
As described in 3, interpolation is performed dimension-wise along the training set, after selecting data center per dimension by manual or automatic (k-means) method.
In our experiment we have selected cross validation [8] over certain fold number of our dataset for assessing the optimal number of centers per dimension, and find those by using k-means [6] algorithm.

For the selected number of extracted centers we add two extreme points for applying proper interpolation for data values beyond limits of the discovered centers.
Now that we have found data centers for each dimension we apply our interpolation method 3 on our data (on both train-test sets) and extract a $\overrightarrow{a} \in \Re^3$ coefficient vector per color patches pair, which is ready to be embedded.

8.1.4 Embedding

Our dataset is now embedded by applying an embedding of the coefficient vectors on a sparse form such:

$\overrightarrow{e} \in \Re^{2\prod_{i=1}^{n} length(\overrightarrow{c_i})}$

where $n = 3$.

Each coefficient vector element is assigned to the exact simplex index in the $\overrightarrow{e}$ embedded vector.

8.1.5 Learning

Learning phase is performed on a linear regression model training process. As described on 5.2, we learn by using **Stochastic Gradient Descent** [9] optimization algorithm, regularized by a known distance such L^1.
Loss function of the training process would be a generalized Mean Average Error [25] loss function, constrained by our semi-metrics critireas for model divergence as described in the learning chapter 5:

$$\vec{w}^{\text{opt}} = \underset{\vec{w}'}{\text{argmin}} \left(\frac{1}{2} ||\vec{w}'||_2^2 + C \sum_{i=1}^{k} \left(d^{\text{reg}}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) + \text{ID}(\overrightarrow{x_{p_{i1}}}, \overrightarrow{x_{p_{i2}}}) \cdot \vec{w}' - y_i \right)^2 \right) \tag{8.1}$$

8.1.6 Assessment

Models assessment is applied by following criteria:

- **MAE** - Mean Absolute Error

$$MAE = \frac{1}{n} \cdot \sum_{i=1}^{n} |y_i^{pred} - y_i^{test}| \tag{8.2}$$

Where n is test set size.

- **MSE** - Mean Square Error

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^{n} (y_i^{pred} - y_i^{test})^2 \tag{8.3}$$

- ΔE - LAB colors distance metrics ΔE metrics applied with MAE data assessment, where basic distance calculation is performed as:

$$\Delta E = \sqrt{(L_1 - L_2)^2 + (A_1 - A_2)^2 + (B_1 - B_2)^2} \tag{8.4}$$

8.2 Results

Let us describe our results being compared to the original paper results as described in 8.5. We display the experiments for each data splitting (camera/color). Our experiments performed by applying cross validation algorithm mentioned at 2. Let us observe the experiments performed on data splits described 8.1.2.

Along all the experiments we perform sweep over the number of centers per dimension $c \in [2, 10]$, which will describe the behavior of all models over datasets' dispersion.
We display several examinations of our results:

- **regularizers scan** - $L1, L2, \Delta E$
- **train .vs. test** sets error
- **unseen colors .vs. unseen cameras**

We also describe a comparison of results based centers extracted by cross validation algo.

8.2.1 Cross-Validation Centers

- Unseen Colors

  Unseen color case was assembled by a certain percentage of all data set as train set, and the rest as test set. In this experiment we have taken the train set size to be 80% of the total data set.

8.2.1.1 Dataset properties

From a total 13500 patches pairs set, the divided train-test sets amounts are:

set	**size**
train	**10800**
test	**2700**

TABLE 8.1: train/set ratio = 4/1

8.2.1.2 Centers locations

Centers locations are shown as extracted via cross-validation 2 for finding optimal number of centers per dimension, and k-means for extracting centers values.
k-means actually found only the inner centers, since the outer centers are fixed bounds.

$$\begin{array}{c}
\begin{array}{ccc} c_0 & c_1 & c_2 \end{array} \\
\begin{pmatrix}
0.0 & 0.0 & 0.0 \\
44.5 & 30.35 & 43.13 \\
84.4 & 80.28 & 83.43 \\
161.59 & 150.96 & 168.81 \\
255.0 & 255.0 & 255.0
\end{pmatrix}
\end{array}$$

8.2.1.3 Experiment Figures

Our model provides the following figures:

MAE	0.38
MSE	0.55
ΔE	0.41

TABLE 8.2: results of cv centers extraction over unseen colors dataset

- Unseen Camera

  Unseen camera scenario was assembled by assigning one of the cameras involved in dataset generation as test set, where the other 3 functions as train set.

8.2.1.4 Dataset properties

From a total 13500 patches pairs set, the divided train-test sets amounts are:

set	**size**	**cameras**
train	**8084**	**Kodak DCS Pro 14n, Konica Minolta Dimage Z3, Nikon Coolpix S6150**
test	**5416**	**Sony DCR-SR32**

TABLE 8.3: train/set ratio = 3/2

8.2.1.5 Centers locations

Centers locations are shown as extracted via cross-validation [8]. for finding optimal number of centers per dimension, and k-means for extracting centers values.
k-means actually found only the inner centers, since the outer centers are fixed bounds.

$$\begin{array}{ccc} c_0 & c_1 & c_2 \end{array}$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 50.6 & 90.55 & 84.43 \\ 101.54 & - & - \\ 171.29 & 160.98 & 158.21 \\ 255.0 & 255.0 & 255.0 \end{pmatrix}$$

As seen in centers list, edges are equal per dimension $[0.0, 255.0]$ for data bounding.

8.2.1.6 Experiment Figures

Our model provides the following figures:

MAE	0.41
MSE	0.50
ΔE	0.49

TABLE 8.4: results of cv centers extraction over unseen cameras dataset

We now display our results along several criteria, as function of number of centers per dimension

8.2.2 Regularizers Comparison

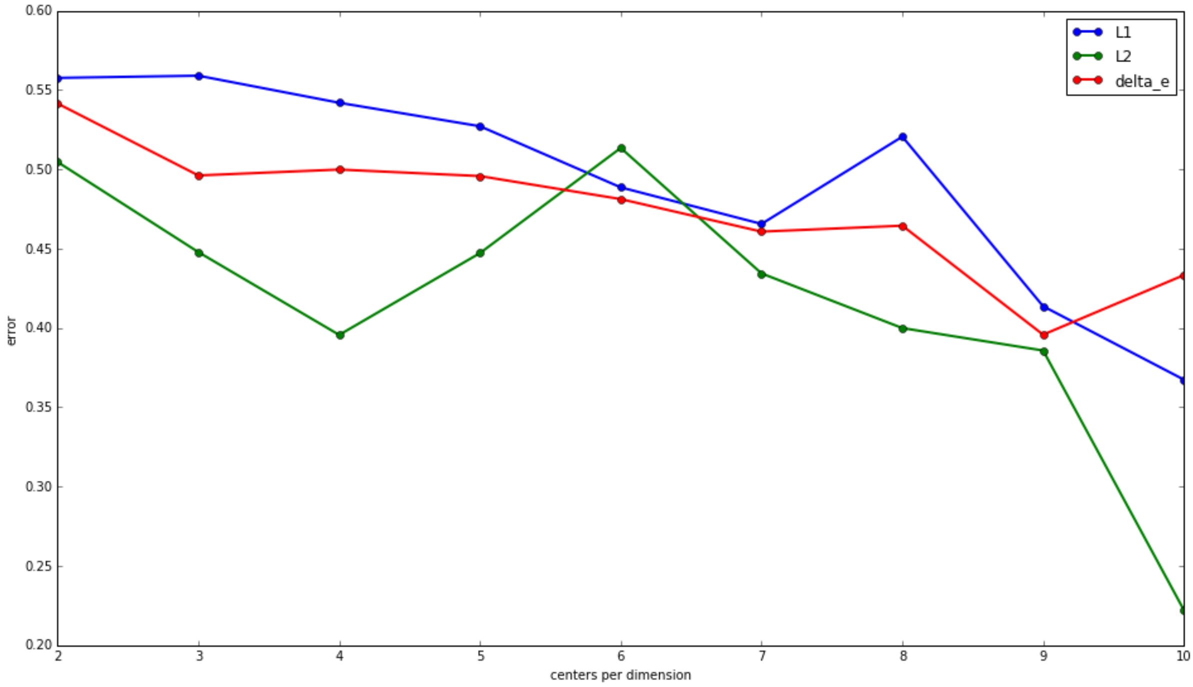


FIGURE 8.2: regularizers error vs centers per dimension

Regularizers comparison 8.2 performs an expected general decreasing trend as we split the data into more segments. Maximal error of any regularizer (0.56) is measured on the minimal data division degree $c = 2$. It is also observed that $L2$ regularizer succeeds applying its penalty on our data in the best manner, where ΔE and $L1$ are manage to penalize dataset.

8.2.3 Train/Test sets Comparison

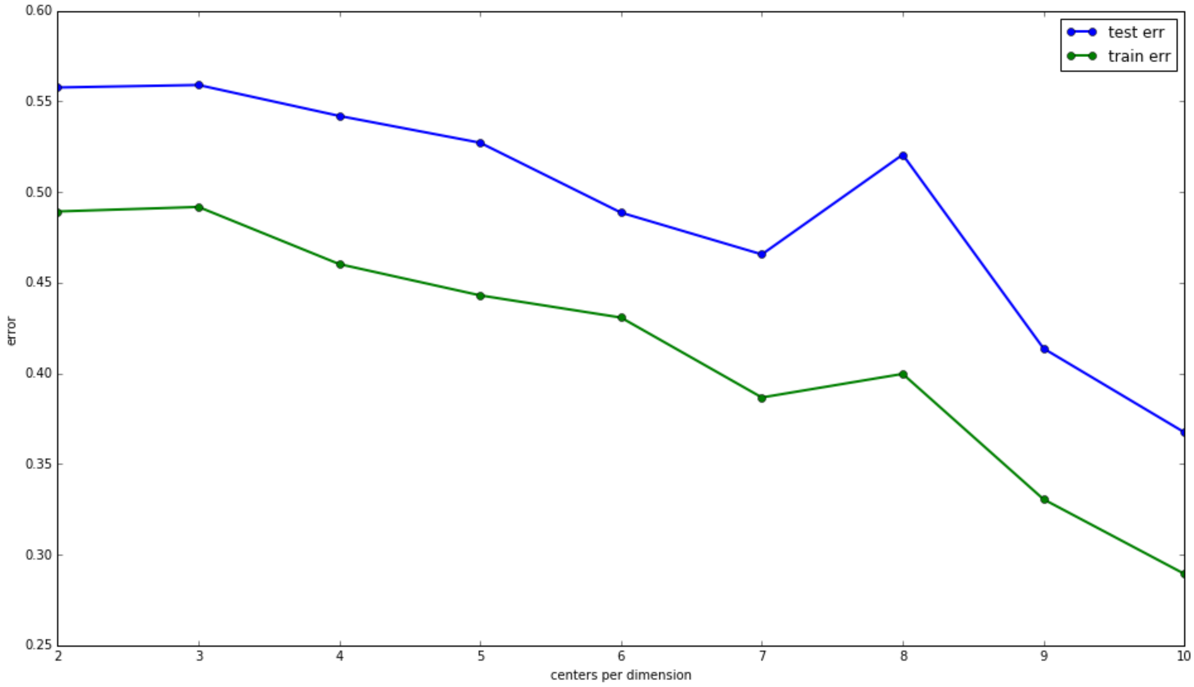


FIGURE 8.3: train and test set errors

Train-test comparison performs an expected general decreasing trend as we split the data into more segments. Maximal error of any set (0.56) is measured on the minimal data division degree - $c = 2$. Similarity of trends behavior as shown in 8.3 indicates well variety of distribution among sets.

8.2.4 Unseen colors/Unseen camera sets Comparison

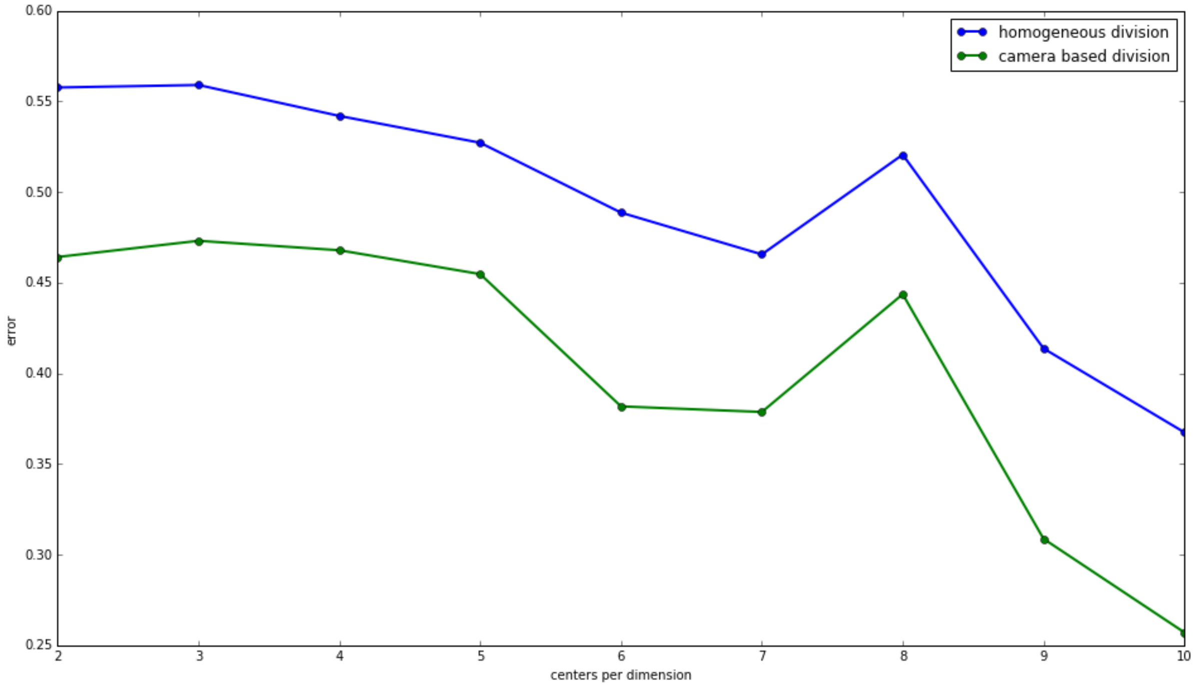


FIGURE 8.4: unseen colors vs unseen camera sets errors

colors-cameras comparison performs an expected general decreasing trend as we split the data into more segments. Maximal error of any set (0.56) is measured on the minimal data division degree - $c = 2$.

8.3 Comparison

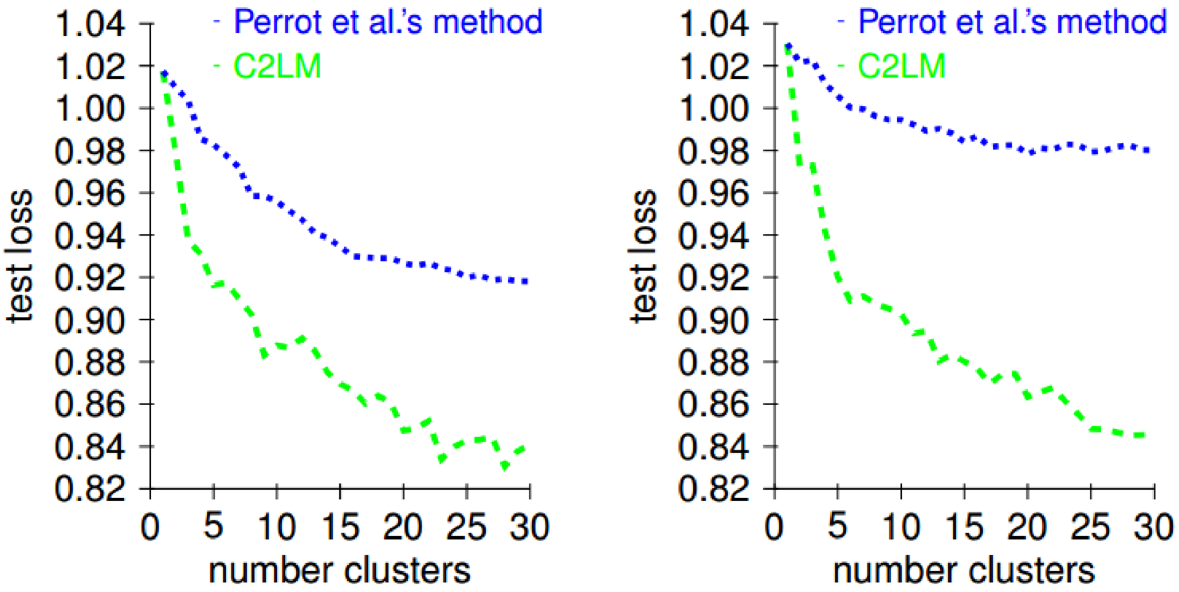


FIGURE 8.5: original paper test loss charts, as function of cluster numbers

Let us treat our results versus the original paper's results.

While [15] (regression) method 8.5 shows results for either camera/color scenarios for model assessment with MAE.
The 8.5 charts' x - axes describes the number of clusters been used in their embedding algorithm.
Minimal results of [15] on both scenarios are **> 0.82 for MAE**, where our results are **< 0.6 for any test case shown above**.
As clearly seen, our results applies the following conditions:

- Outperforming reference original paper error scores
- Overfitting avoidance example method of applying cross validated centers based on our training set solely.

This information is a meaningful confirmation of the assumption that our method is valid to work with in terms of reliability. In the following chapter we will discuss the further explorations should be performed to examine the rest of performance criterias mentioned above.

Chapter 9

Conclusions and discussion

9.1 Summary

In this work we have proposed a new embedding method for a single vector and for a pair of vectors. This embedding method enables:

1. efficient classification and regression of functions of single vectors
2. efficient approximation of distance functions
3. general, non-Euclidean, semimetric learning

. For the best of our knowledge, this is the first work that enables learning any general, non-Euclidean, semimetrics.
That is, our method is a universal semimetric learning and approximation method that can approximate any distance function with as high accuracy as needed with or without semimetric constraints.
In this work, we have displayed a detailed exploration of our ID method, describing its basic classification/regression algorithmic flow for objects embedding and its applications options.

We have described how it may be applied on pair objects matters, such similarity/dissimilarity problems 3.

Also described how to apply our method on single objects embedding problems, for cases such classification/regression problems 4.

On chapter 5, the learning phase of the embedding method was described, detailing a specific adaptation to SGD [9] algorithm in order to fit our data types to optimize the optimization process of loss convergence.

Chapter 6 handles the time complexity issues related to the embedded objects. Each step in the process is analyzed in terms of current complexity analysis of the general method.

Chapter 7 handles the memory issues related to the embedded objects, which are originally sparsed, and how it may be treated in order to save memory.

Chapter 8 examines our IDD method on e test case displayed by [15], which studies method to observe perceptual colors difference. Here we have validated the correctness of our method by displaying out-scoring results in comparison to original articles results.

9.2 further exploration

This work displays a continuous non-linear embedding method for any desired classification/regression task. In chapter 8 we displayed an accuracy exploration of our method, on the object pairs embedding scenario.

In addition to this examination, the following objectives may be observed:

- **various centers per dimension** may be a good valuation of the optimal condition per problem, in case cross validation may treat single dimension at a time. By optimizing several dimensions in one shot, one can decide of a fixed number of centers for all dimensions, which may be still optimized in matters of accuracy, time and memory.
- **Memory** consumption may be observed among various embedding method include ours, while using our memory saving tips at chapter 6, where in our method (and in [15], number of centers/clusters is a valid argument to parameterize in this kind of comparison)
- **time** complexity may also taken in comparison among various object-embedding methods, while using our chapter 7 theories.
- **end-to-end application embedding** such image segmentation as shown in [15], may assist evaluating this method
- **single objects embedding** may also be inspected for some classification/regression task, again with all parameters mentioned above (accuracy, time, memory)

Bibliography

[1] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.

[2] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE, 2005.

[3] Richard Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.

[4] Per-Erik Danielsson. Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248, 1980.

[5] Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L Massart. The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18, 2000.

[6] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[7] Charles AR Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[8] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA, 1995.

[9] Bottou L. Large-scale machine learning with stochastic gradient descent. *Springer*, 2010.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[11] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[12] Joseph L Mundy and Andrew Zisserman. Appendix-projective geometry for machine vision, 1992.

[13] Dorothy Nickerson, Walter C Granville, and J Opt. The farnsworth-munsell 100-hue and dichotomous tests for color vision. *J. opt*, 1943.

[14] Ofir Pele. *Distance Functions: Theory, Algorithms and Applications*. PhD thesis, The Hebrew University of Jerusalem, 2011.

[15] Muselet D. Perrot M., Habrard A. Modeling perceptual color differences by local metric learning. *Springer*, 2014.

[16] Tim Peters. Timsort, 2002.

[17] Shai Shalev-Shwartz, Yoram Singer, and Andrew Y Ng. Online and batch learning of pseudo-metrics. In *Proceedings of the twenty-first international conference on Machine learning*, page 94. ACM, 2004.

[18] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, pages 807–814. ACM, 2007.

[19] Manfred J Sippl and Harold A Scheraga. Cayley-menger coordinates. *Proceedings of the National Academy of Sciences*, 83(8):2283–2287, 1986.

[20] Shamik Sural, Gang Qian, and Sakti Pramanik. Segmentation and histogram generation using the hsv color space for image retrieval. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 2, pages II–II. IEEE, 2002.

[21] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[22] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.

[23] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[24] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009.

[25] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[26] Valentina Zantedeschi, Remi Emonet, and Marc Sebban. Metric learning as convex combinations of local models with generalization guarantees. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[27] Xuemei Zhang and Brian A Wandell. A spatial extension of cielab for digital color-image reproduction. *Journal of the Society for Information Display*, 5(1):61–63, 1997.