

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

September 2, 2016



## Declaration of Authorship

I, Yakir BEN-ALIZ, declare that this thesis titled, “Multidimensional Interpolated Discretized for Objects and Object Pairs Embedding” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

---

Date:

---



*“Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism.”*

Dave Barry



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  $\chi$ ). 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

<b>Declaration of Authorship</b>	<b>iii</b>
<b>Abstract</b>	<b>vii</b>
<b>Acknowledgements</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Metrics	1
1.1.1 Definition	1
1.1.2 Metrics Properties	2
1.1.2.1 Non-negativity	2
1.1.2.2 Identity of Indiscernibles	2
1.1.2.3 Symmetry	2
1.1.2.4 Sub-additivity (Triangle Inequality)	2
1.1.3 Semi-metrics	2
1.1.4 Pseudo-metrics	2
1.2 Metric Learning	3
1.3 Bin-to-Bin & Cross-Bin Metrics	3
1.4 Mahalanobis Distance	3
1.5 Related Work	3
1.6 Contribution	4
<b>2 ID Embedding</b>	<b>5</b>
2.1 Discretization	5
2.2 Interpolation	5
2.3 Assigning	5
<b>3 Interpolated Discretized object pairs embedding</b>	<b>7</b>
3.1 1D case	7
3.1.1 Discretization	7
3.1.2 Interpolation	8
3.1.2.1 Interpolation Coefficients extraction - 1D	8
3.1.3 Assigning	9
3.2 multidimensional case	9
3.2.1 Definition	9
3.2.2 Discretization	9
3.2.3 Interpolation	9
3.2.3.1 find bounding hypercube	10
3.2.3.2 find bounding simplex	10
3.2.3.3 calculate sub-volumes of all simplices involved	10
3.2.3.4 Simplex Volume Calculation	10
3.2.3.5 Efficient method - calculating Barycentric Coordinates (BCC)	11
3.2.3.6 O(n) solution	11
3.2.3.7 Methods identity for n=2	11

3.2.3.8	Volumes method . . . . .	12
3.2.3.9	Recursive method . . . . .	13
3.2.4	Assigning . . . . .	14
3.3	Non-euclidean non-embeddable Metrics example . . . . .	15
<b>4</b>	<b>Interpolated Discretized Single objects Embedding</b>	<b>19</b>
4.1	Discretization . . . . .	19
4.2	Interpolation . . . . .	19
4.3	Assigning . . . . .	20
<b>5</b>	<b>Learning</b>	<b>21</b>
5.1	Learning the Classification Function . . . . .	21
5.1.1	Efficient Stochastic Gradient Descent . . . . .	22
<b>6</b>	<b>Time Complexity</b>	<b>23</b>
6.1	Discretization . . . . .	23
6.2	Interpolation . . . . .	23
6.3	Find the bounding hypercube . . . . .	23
6.4	Find the bounding simplex . . . . .	24
6.5	ID coefficients extraction . . . . .	24
6.6	Assigning . . . . .	24
<b>7</b>	<b>Memory Complexity</b>	<b>25</b>
7.1	Definition . . . . .	25
7.1.1	SIFT Example . . . . .	25
<b>8</b>	<b>Experiments</b>	<b>27</b>
<b>9</b>	<b>Conclusions and discussion</b>	<b>29</b>
<b>A</b>	<b>Appendix Title Here</b>	<b>31</b>
	<b>Bibliography</b>	<b>33</b>

# List of Figures

3.1	discretization matrix	7
3.2	1D data sample coefficients calculation matrix	16
3.3	flattening 2d matrix	17
3.4	triangle non-embeddable dataset	17
4.1	discretization matrix	19
7.1	SIFT algorithm used for template matching	26
7.2	SIFT kp descriptor example	26



# List of Tables

3.1 higher dimensions' number of simplices and vertices . . . . .	13
---	----





# List of Algorithms

1	Embedding Method - General . . . . .	6
2	Embedding Method for ID N-Dimensional Pairs dataset . . . . .	14
3	Embedding Method for ID N-Dimensional single vectors dataset . . . . .	20



# List of Abbreviations

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



# List of Symbols

$a$	distance	m
$P$	power	W (J s <sup>-1</sup> )
$\omega$	angular frequency	rad



*/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[] (such as  $L_2$  norm, representing the euclidean distance []), or can be learned from datasets - metric learning.

In metric learning, most of the works learn a Mahalanobis distance. These methods [1-12] 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 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  $\chi$ ). 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.

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 [] 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 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:

$$d : q_1 \times q_2 \rightarrow \mathbb{R} \quad (1.1)$$

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

$$d(q_1, q_2) \geq 0 \quad (1.2)$$

#### 1.1.2.2 Identity of Indiscernibles

$$d(q_1, q_2) = 0 \iff q_1 = q_2 \quad (1.3)$$

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

$$d(q_1, q_2) = d(q_2, q_1) \quad (1.4)$$

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)

$$d(q_1, q_3) \leq d(q_1, q_2) + d(q_2, q_3) \quad (1.5)$$

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:

$$q_1 = q_2 \Rightarrow d(q_1, q_2) = 0 \quad (1.6)$$

## 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[1]. 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[1] and  $\chi^2$ [1]).

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  $\chi^2$  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  $A \in \mathbb{R}^{N \times N}$  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 :

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

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-semidefinite (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:

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

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 [1] 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. [3] proposed to learn a convolutional neural net as a nonlinear transformation before applying the  $\ell_2$  norm. They showed excellent results on image data. Babenko et al. [12] 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 non-linear 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  $\ell_2$  distance. A QC distance have parameters that can be learned. However, a serious limitation is that it can only model  $\ell_2$ -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[1]. 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  $\vec{v} \in \mathbb{R}^{10^9}$  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  $\vec{v} \in \mathbb{R}^{C_i}$  vector.

Any common clustering method may benefit in this, with one exception: The extremum points of the sorted discretized vector must surround the extremum 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 (sparsed) vector.

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.

---

**Algorithm 1** Embedding Method - General
 

---

**Input:**  $L$  sized, vectorized  $n$ -dimensional dataset

**Input:** number of centers per dimension -  $C$

**Output:**  $\vec{\phi}$ :  $L$  sized set, embedded, sparse vectors

Find centers vectors

$V$  shall be a set of centers vectors

**for all**  $dim$  in  $n$  **do**

$V_{dim} \leftarrow centers \ vector \ per \ dim$

**end for**

Find embedded coefficients for all dataset

$\vec{\phi} = C^n$  length empty  $\vec{\phi}$  embedded vectors

**for all**  $vec$  in  $L$  **do**

find  $vec$  bounding hypercube

find  $vec$  bounding simplex (permutation method)

$\vec{\lambda} \leftarrow$  find  $vec$  barycentric coefficients

$\vec{\lambda} \leftarrow$  normalize( $\vec{\lambda}$ )

**end for**

Assign

**for all**  $\vec{vec}$  in  $emb - set$  **do**

$inds \leftarrow$  find vertices from hypercube and simplex locations

**for all**  $i$  in  $inds$  **do**

$\vec{vec}(i) \leftarrow \vec{\lambda}(j(i))$  -  $j$  is the assigning function between the coef. vector and embedding vector

**end for**

**end for**

**return**  $\vec{\phi}$

---

## 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[], 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:

$$ID : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^d \quad (3.1)$$

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

$$d(\vec{x}_1, \vec{x}_2) = ID(\vec{x}_1, \vec{x}_2) \times \vec{w} \quad (3.2)$$

this embedding shall obtain semimetric constraints applied.

#### 3.1.1 Discretization

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:  $\vec{c} \in \mathbb{R}^{C_i}$

$$W = \begin{pmatrix} c_{2(1)} \\ \vdots \\ c_{2(m)} \end{pmatrix} \begin{pmatrix} s_{1,1} & \dots & \dots & \dots & s_{1,m} \\ \vdots & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ \vdots & & & & \vdots \\ s_{n,1} & \dots & \dots & \dots & s_{n,m} \end{pmatrix}$$

FIGURE 3.1: discretization matrix  $W$  as built from 2 vectors  $c_1, c_2$ . this matrix is 2D as it is an outcome of 2 - 1D vectors discretization processes

vector. The vector's pair defines the shape of a distance matrix  $W \in \mathbb{R}^{C_i \times C_j}$ , where each element  $W_{i,j}$  describes the distance between two cluster centers form vectors -  $v_a, v_b$

### 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:

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

where:

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

Please notice that there are only 4 elements different than zero at  $\alpha$ , so this expression represents the non-zero elements only provides value to 1DIDD expression

$a, b$  parameterizations are described as:

- $$a_{(1)} = a_{(2)} = \operatorname{argmax}_{(c_i)} \{v_{c_i} \leq x_i\} \quad (3.4)$$

- $$a_{(3)} = a_{(4)} = \operatorname{argmin}_{(c_i)} \{v_{c_i} \geq x_i\} \quad (3.5)$$

- $$b_{(1)} = b_{(3)} = \operatorname{argmax}_{(c_j)} \{v_{c_j} \leq x_j\} \quad (3.6)$$

- $$b_{(2)} = b_{(4)} = \operatorname{argmin}_{(c_j)} \{v_{c_j} \geq x_j\} \quad (3.7)$$

#### 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 assembled from the clusters vectors per feature. this square is divided into 2 triangles (simplices) along its main diagonal, as described in figure ??.

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 ??.



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 ??.

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 [] 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:

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

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.

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  $x_1, x_2 \in \mathbb{R}^n$ .

*NDIDD* embedding handles this pair as a **joint  $2n$  dimensional vector**, flatten in the following order:

$$\vec{p} = [x_1^1, x_2^1, \dots, x_1^n, x_2^n], \quad \vec{p} \in \mathbb{R}^{2n} \quad (3.9)$$

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

### 3.2.3.1 find bounding hypercube

First, we place the sampled vector  $\vec{p}$  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  $\vec{p}$ . 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:

$$0 \leq p_{t(1)} \leq p_{t(1)} \leq \dots \leq p_{t(2n-1)} \leq p_{t(2n)} \leq 1 \quad (3.10)$$

<http://www.mathpages.com/home/kmath664/kmath664.htm>

where:

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

so how we select the right simplex (and permutation) for a given point  $\vec{p}$ ? Each point  $\vec{p}$  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  $\vec{p}$ , we calculate the volumes of  $T$  sub-simplices, so every simplex is assembled from  $T - 1$  vertices plus  $\vec{p}$ .

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:

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

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

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

### 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  $\vec{p}$ . Given a point  $\vec{p} \in \mathbb{R}^{2n}$ , 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:

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

so the equation we shall use is this:

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

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

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

and the remained coefficient extracted from the final step:

$$\lambda_0 = 1 - \sum_{j=1}^{2n} \lambda_j \quad (3.15)$$

### 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  $\vec{p} = [p_1, p_2, p_3, p_4]$ , 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  $\lambda_4$  calculation, since it is straightforward from matrix's gradation

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

for  $\lambda_3$  we use the proceed the algorithm:

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

$\lambda_2$ :

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

$\lambda_1$ :

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

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

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

(3.19)

for  $\lambda_0$  we apply the final step expression:

$$\begin{aligned}
 \lambda_0 &= V_{[p, r^1, r^2, r^3, r^4]} = \begin{vmatrix} 1 & p_1 & p_2 & p_3 & p_4 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{vmatrix} \\
 &= 1 - p_1 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} + p_2 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} - p_3 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} + p_4 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} \\
 &= 1 - p_4 - p_3 = 1 - (\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4)
 \end{aligned}$$

(3.20)

final coefficient vector produced by volumes method would be:

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

### 3.2.3.9 Recursive method

based on the formulation developed above 3.14, we extract the now extract the  $\lambda$  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  $\lambda_0$  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  $\vec{p}$ .

–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  $\alpha$  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  $\alpha$  vector and a learned weights vector (equivalent to the  $W$  matrix from the  $1D$  scenario).

---

#### Algorithm 2 Embedding Method for ID N-Dimensional Pairs dataset

---

**Input:**  $L$  sized, vectorized  $n$ -dimensional pairs dataset

**Input:** set of centers per dimension -  $C$

**Output:**  $\vec{\phi}$ :  $L$  sized set, embedded, sparse vectors

**Find centers vectors**

$V$  shall be a set of centers - vectors

**for all**  $dim$  **in**  $n$  **do**

$V_{dim} \leftarrow centers \text{ vector per } dim$

**end for**

**Find embedded coefficients for all dataset**

$\vec{\phi} = C^{2n}$  length empty  $\vec{\phi}$  embedded vectors

concat all pairs into a single vector  $\vec{p}$

**for all**  $\vec{p}$  **in**  $L$  **do**

find  $\vec{p}$  bounding hypercube

find  $\vec{p}$  bounding simplex (permutation method)

$\vec{\lambda} \leftarrow \text{find } \vec{p} \text{ barycentric coefficients}$

$\vec{\lambda} \leftarrow \text{normalize}(\vec{\lambda})$

**end for**

**Assign**

**for all**  $\vec{\phi}$  **in**  $emb - set$  **do**

$inds \leftarrow \text{find vertices from hypercube and simplex locations}$

**for all**  $i$  **in**  $inds$  **do**

$\vec{\phi}_{(i)} \leftarrow \vec{\lambda}(j(i))$  -  $j$  is the assigning function between the coef. vector and embedding vector

**end for**

**end for**

**return**  $\vec{\phi}$

---

### 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:

$$f : \{\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{x}_4\} \rightarrow \mathbb{R}^k \quad (3.22)$$

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

As triangle-inequality is tight for:  $\vec{x}_4, \vec{x}_1, \vec{x}_2$ ,

$f(\vec{x}_4), f(\vec{x}_1), f(\vec{x}_2)$  are collinear in  $\mathbb{R}^k$

From set formation symmetry property, the same colinearity applies on:

$f(\vec{x}_3), f(\vec{x}_1), f(\vec{x}_2)$

This common colinearity of those tuples leads to the following equation:

$$\|f(\vec{x}_4) - f(\vec{x}_3)\|_2 = 0$$

But this fact is contradicting that

$$d(\vec{x}_4, \vec{x}_3) = 2$$

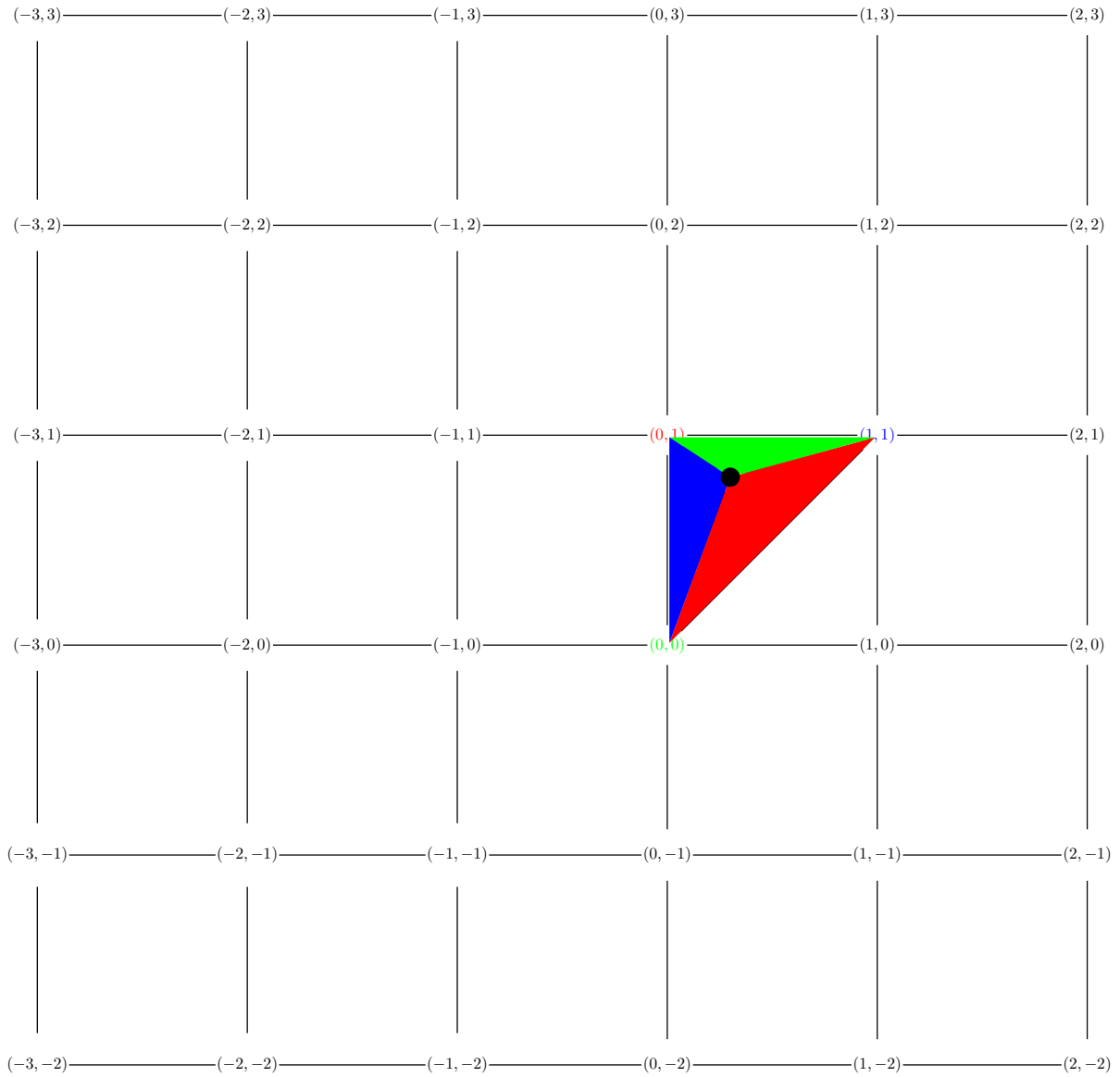


FIGURE 3.2: this matrix scheme describes a given data sample coefficients calculation according to its containing hypercube and simplex. The examined data sample is  $(0.3, 0.8)$ , so its containing hypercube would be the cell between  $[0, 1]$ . In this hypercube we search the containing simplex (triangle 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.



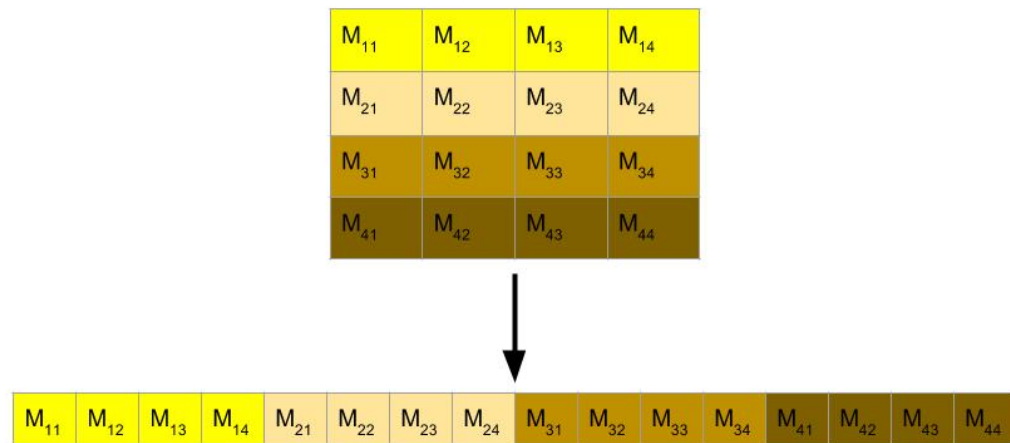


FIGURE 3.3: flattening 2d matrix to row-wise 1d vector

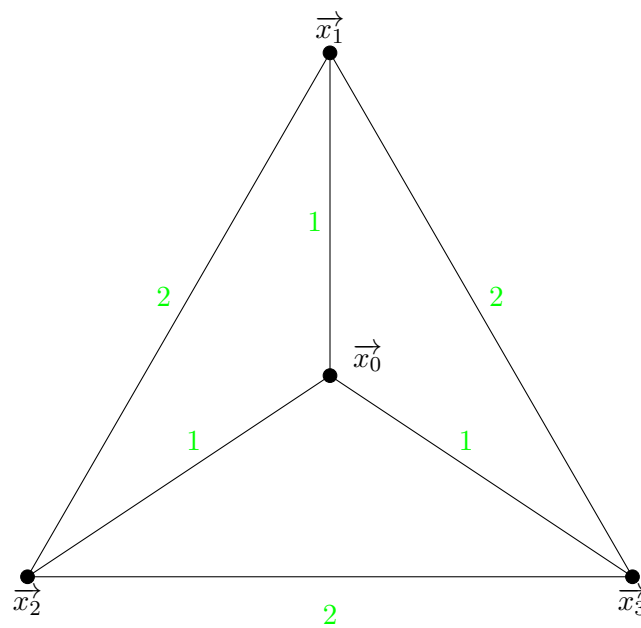


FIGURE 3.4: this is a scheme of a non-embeddable dataset



## 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 ??).

### 4.2 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

$$\begin{pmatrix} (v_1, v_2, \dots, v_{n-1}, v_n) \\ m_{1,1} & \dots & \dots & \dots & m_{1,n} \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ \vdots & & & & \ddots & \vdots \\ m_{C-1,1} & & & & & m_{C-1,n} \\ m_{C,1} & \dots & \dots & \dots & \dots & m_{C,n} \end{pmatrix}$$

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 3** Embedding Method for ID N-Dimensional single vectors dataset

---

**Input:**  $L$  sized, vectorized n-dimensional dataset

**Input:** set of centers per dimension -  $C$

**Output:**  $\vec{\phi}$ :  $L$  sized set, embedded, sparse vectors

**Find centers vectors**

$V$  shall be a set of centers - vectors

**for all** dim in  $n$  **do**

$V_{dim} \leftarrow \text{centers vector per dim}$

**end for**

**Find embedded coefficients for all dataset**

$\vec{\phi} = C^n$  length empty  $\vec{\phi}$  embedded vectors

**for all**  $\vec{p}$  in  $L$  **do**

find  $\vec{p}$  bounding hypercube

find  $\vec{p}$  bounding simplex (permutation method)

$\vec{\lambda} \leftarrow \text{find } \vec{p} \text{ barycentric coefficients}$

$\vec{\lambda} \leftarrow \text{normalize}(\vec{\lambda})$

**end for**

**Assign**

**for all**  $\vec{\phi}$  in  $emb - set$  **do**

$inds \leftarrow \text{find vertices from hypercube and simplex locations}$

**for all**  $i$  in  $inds$  **do**

$\vec{\phi}_{(i)} \leftarrow \vec{\lambda}(j(i))$  -  $j$  is the assigning function between the coef. vector and embedding vector

**end for**

**end for**

**return**  $\vec{\phi}$

---

## 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 the 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:

$$P = \begin{bmatrix} p_{11} & p_{12} \\ \vdots & \vdots \\ p_{i1} & p_{i2} \\ \vdots & \vdots \\ p_{k1} & p_{k2} \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_k \end{bmatrix} \quad (5.1)$$

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

$$y_i = \begin{cases} -1 & \text{if } \vec{x}_{p_{i1}} \text{ and } \vec{x}_{p_{i2}} \text{ are similar} \\ +1 & \text{if } 2; S = [ID(\vec{x}_{p_{i1}} \text{ and } 1[S_1], \vec{x}_{p_{i2}} \text{ are non - similar} \end{cases} \quad (5.2)$$

We define a classification (similarity) function:

$$similar(\vec{x}_1, [\vec{x}_2]) = \begin{cases} -1 & \text{if } d(\vec{x}_1, \vec{x}_2) = IDD(\vec{x}_1, \vec{x}_2) \cdot \vec{w} < t \\ +1 & \text{otherwise} \end{cases} \quad (5.3)$$

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 [80], 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 the 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*) 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 [67], we can learn our weights (including  $t$  parameter):

$$\vec{w}^{opt}, t^{opt} = \underset{\vec{w}, t}{\operatorname{argmin}} \left( \frac{1}{2} \|\vec{w} - \vec{w}^{reg}\|_2^2 + C \sum_{i=1}^k \max(1 - (IDD(\vec{x}_{p_i1}, \vec{x}_{p_i2}) \cdot \vec{w} - t)y_i, 0) \right) \quad (5.4)$$

Where:

- $\vec{w}^{reg}$  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:

$$\frac{\partial \frac{1}{2} \cdot \frac{1}{k} \cdot \|\vec{w} - \vec{w}^{reg}\|_2^2}{\partial \vec{w}} = \frac{1}{k} (\vec{w} - \vec{w}^{reg}) \quad (5.5)$$

We can use an **overcomplete** representation of the weights vector  $\vec{w}$  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[]):

$$\vec{w} = \beta \cdot \hat{\vec{w}} + \gamma \cdot \vec{w}^{reg} \quad (5.6)$$

So instead of the common *SGD* weights update:

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

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

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

Which allows to separate both coefficients of the overcomplete representation:

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

## 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 ()

### 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. We will try to refer to  $k - means$  ?? based methods. Though the problem of finding the global optimum of the  $k - means$  objective function:

$$arg \min_s \sum_{i=1}^k \sum_{x_j \in S_i} \|x_j - \mu_i\|^2 \quad (6.1)$$

is NP-hard[].

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

$$O(iknd)$$

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(), we divide the interpolation phase time complexity for several phases:

### 6.3 Find the bounding hypercube

Finding a given vector's bounding box is actually finding 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:

$$O(n \log(n))$$

## 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:

$O(n \log(n))$

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

## 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:

$$O(n \log(n)) + O(n \log(n)) + O(n) = O(n \log(n))$$

## 6.6 Assigning

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



## 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:

$$ID(\vec{x}_1, \vec{x}_2; S) = [ID(\vec{x}_1[S_1], \vec{x}_2[S_1]), \dots, ID(\vec{x}_1[S_g], \vec{x}_2[S_g])] \quad (7.1)$$

where  $S$  is the defined sub-domain group:

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

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

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

#### 7.1.1 SIFT Example

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

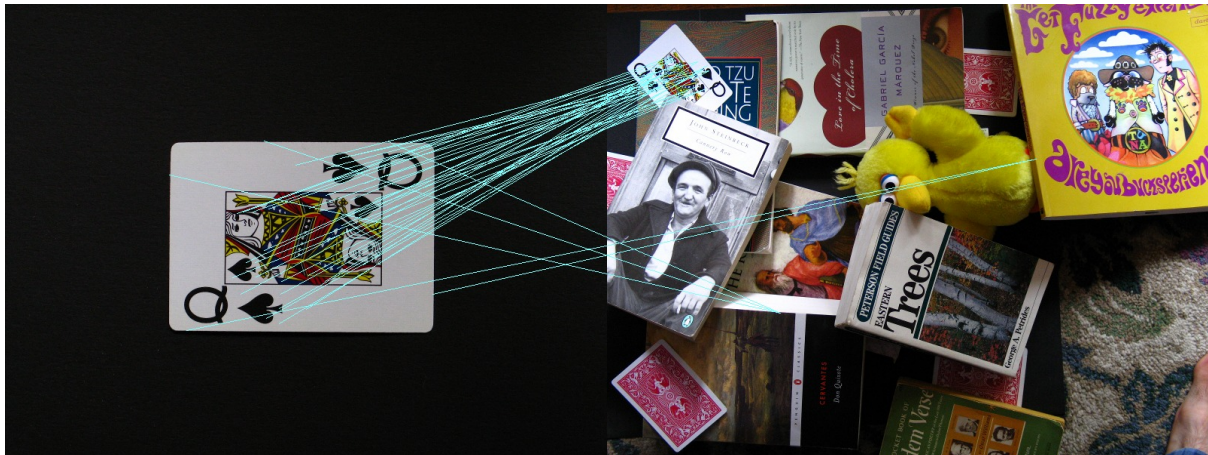


FIGURE 7.1: SIFT algorithm used for template matching. In this example each image is scanned and a set of keypoints (a significant point) is found, and for each keypoints a 3D features vector is calculated. These vectors are being compared between the images, and a similarity is measured among those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

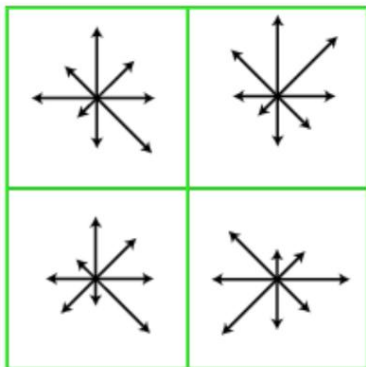


FIGURE 7.2: SIFT keypoint descriptor scheme. A  $16 \times 16$  neighborhood around the keypoint is taken. It is divided into 16 sub-blocks of  $4 \times 4$  size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to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

## Chapter 8

# Experiments



## **Chapter 9**

# **Conclusions and discussion**



## **Appendix A**

### **Appendix Title Here**

Write your Appendix content here. blablabla





# Bibliography

- Arnold, A. S. et al. (1998). "A Simple Extended-Cavity Diode Laser". In: *Review of Scientific Instruments* 69.3, pp. 1236–1239. URL: <http://link.aip.org/link/?RSI/69/1236/1>.
- Hawthorn, C. J., K. P. Weber, and R. E. Scholten (2001). "Littrow Configuration Tunable External Cavity Diode Laser with Fixed Direction Output Beam". In: *Review of Scientific Instruments* 72.12, pp. 4477–4479. URL: <http://link.aip.org/link/?RSI/72/4477/1>.
- Wieman, Carl E. and Leo Hollberg (1991). "Using Diode Lasers for Atomic Physics". In: *Review of Scientific Instruments* 62.1, pp. 1–20. URL: <http://link.aip.org/link/?RSI/62/1/1>.