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 20, 2016

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[] (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 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.

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:

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

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 [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 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 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[]. 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 computated.

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 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: $\overrightarrow{\phi}$: 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 $\overrightarrow{\phi}$

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: **einstein**

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

, 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

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

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

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

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 **??**.

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:

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

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.

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:

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

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 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:

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

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:

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

$$= \mathrm{p}_1 \cdot \begin{vmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} - p_2 \cdot \begin{vmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} + p_3 \cdot \begin{vmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} - p_4 \cdot \begin{vmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

$$= \mathrm{p}_4 - p_3 = p_4 - (\lambda_2 + \lambda_3 + \lambda_4)$$

(3.19)

for λ_0 we apply the final step expression:

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

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

$$= 1 - p_1 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} + p_2 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} - p_3 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} + p_4 \cdot \begin{vmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} \tag{3.20}$$

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

final coefficient vector produced by volumes method would be:

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

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 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: $\overrightarrow{\phi}$: 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
$\overrightarrow{\tilde{\lambda}} \leftarrow$ 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**
$\overrightarrow{\phi}_{(i)} \leftarrow \overrightarrow{\tilde{\lambda}}(j(i))$ – 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 colinearity applies on:
$f(\overrightarrow{x_3}), f(\overrightarrow{x_1}), f(\overrightarrow{x_2})$

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

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

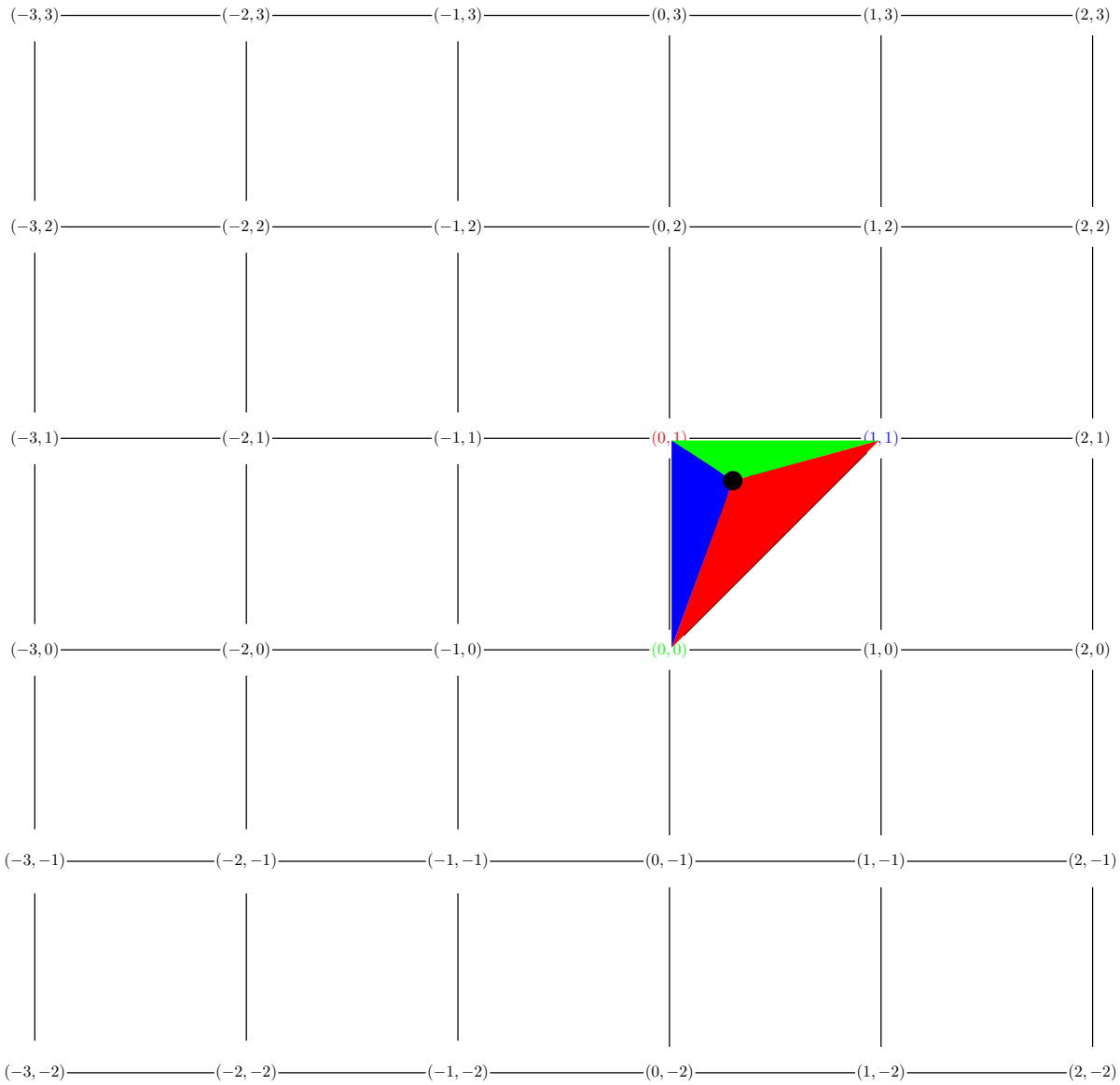

FIGURE 3.1: this matrix scheme describes a given data sample coefficients calculation according to its containing hypercube and simplex.The examined data sample is $(0.3, 0.8)$, so its containing hypercube would be the cell between $[0, 1]$. In this hypercube we search the containing simplex (triangle 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.

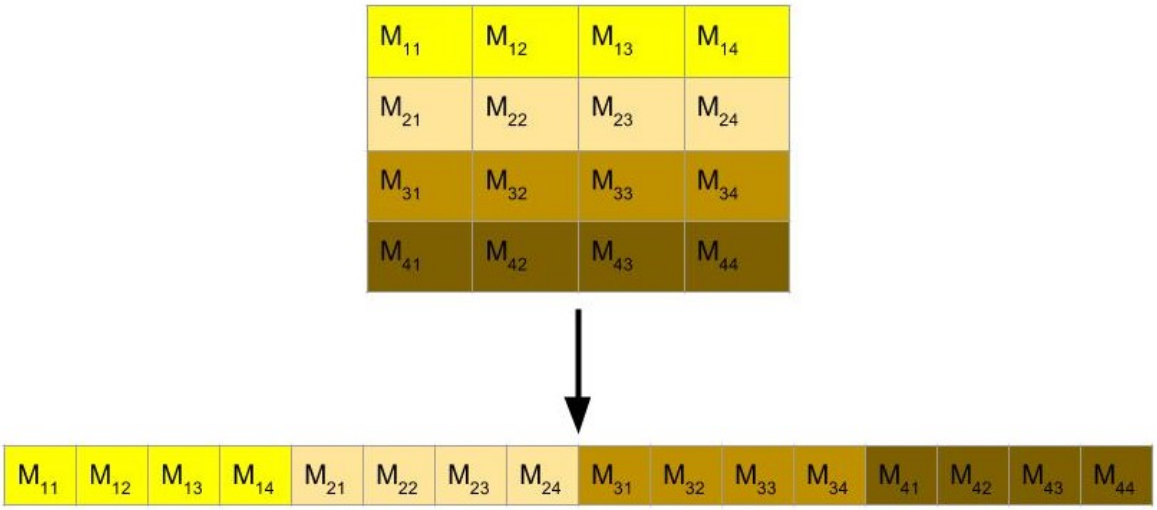

FIGURE 3.2: flattening 2d matrix to row-wise 1d vector

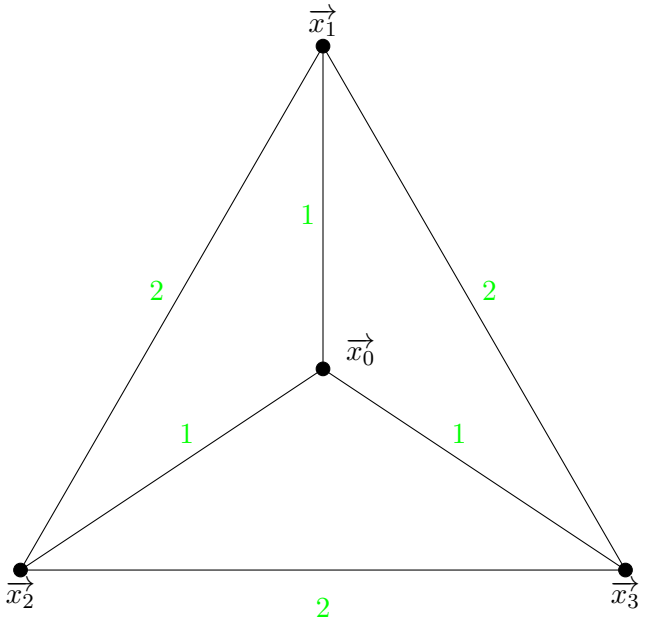

FIGURE 3.3: non-embeddable dataset scheme, describing a 2D triangled set, where 4 centers represents 4 data groups which their average distances shown in green

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 **??**.

4.2 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

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

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 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: $\overrightarrow{\phi}$: 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 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:

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

Where P set refers to X set indices and tghsgfds 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 [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 ther 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):

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

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:

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

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[]):

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

So instead of the common SGD weights update:

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

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

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

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:

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

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(), 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:
$O(nlog(n))$

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[]/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:

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

6.6 Assigning

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

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[]** 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), $n = 128$, 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.

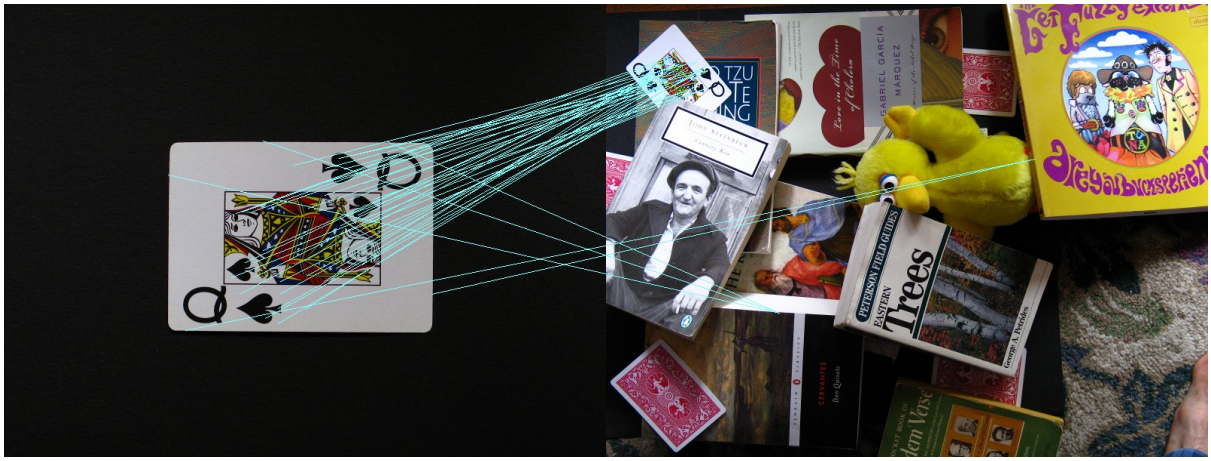

FIGURE 7.1: SIFT algorithm used for template matching. In this example each image is scanned and a set of keypoints (a significant point) is found, and for each keypoints a 3D features vector is calculated. These vectors are being compared between the images, and a similarity is measured among those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

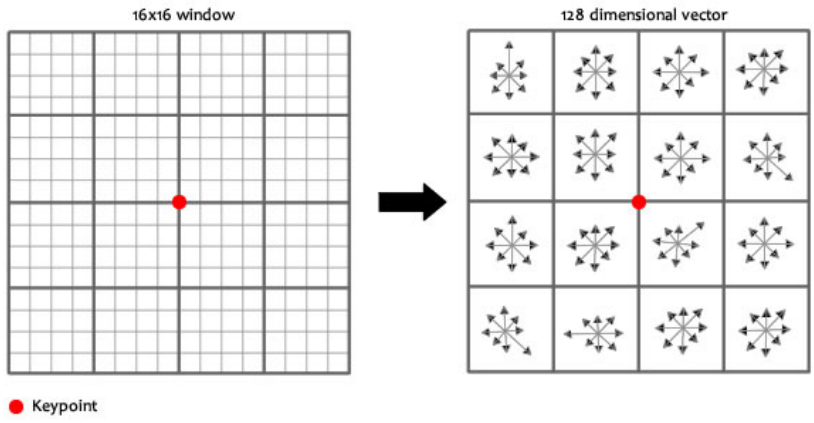

FIGURE 7.2: SIFT keypoint descriptor scheme. A 16x16 neighborhood around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor.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

Chapter 8

Experiments

In this chapter we will display some novel usage in the ID functionalities and qualities.

Here we will use our ID method in order to analyze a problem called image retargeting which is described in blabla's work ID method will apply as a similarity metrics calculator. At last, we will compare its objective images ranking with users ranking the retargeting methods.

single ID embedding 3 will be demonstrated by image classification task

For this section we have used an "image retargeting" dataset **ggg** which contains a 91 various images dataset, each image has several attributes such lines/edges, symmetry, faces etc. In this dataset, each several retargeting methods have been applied on every image.

"http://people.csail.mit.edu/mrub/retargetme/"

FIGURE 8.1: Example of retargeting the butterfly image to half its size. In this study we evaluate 8 different image retargeting methods, asking users to compare their results and examine what qualities in retargeted images mattered to them. We also correlate the users' preferences with automatic image similarity measures. Our findings provide insights on the retargeting problem, and present a clear benchmark for future research in the field.

8.1 experiments procedure

Every image has 8 various retargeting methods. For each image on dataset we have obtained a single vector descriptor. This descriptor is a SIFT descriptor, calculated on the entire image from its center as shown in figure **??**.

Our set is a $n = 128$ length dataset, divided to 37 classes.
As described in 3, for relatively long vectors such SIFT descriptors, ID method requires memory saving methods. Such method has described in **??** as **grouping** method. In this method we sample sub-domains from the datasets' vectors, run the ID method separately on each sub-domain, then concatenate the embedded sub-vectors to a flatten embedded vector.

The original experiment in the retargeting paper, dealt with ranking comparison, between human perspective, and an objective metrics one. Each metric has ordered its similarity results

FIGURE 8.2: Original butterfly image from retargeting dataset, with an overlay SIFT descriptor on it, as we have measured on every image in the dataset. All SIFT descriptors were taken from the center of the image. Their radius was as the shorter image-side, and their orientation was $+90°$.

by ascending order (as described in **gfdfvdfs** user votes was sorted by descending order so it may be equivalent to similarity - order).

Correlation was calculated by Kendall-tau correlation distance between users votes and objective metrics similarity ranking results.

8.2 Similarity with ID

Here we have obtained the single objects embedding ID method for similarity analysis. This was performed by applying a SVR **SVR** classifier on embedded single vectors. SVR model trained on the 37 reference images, which function as this problem's training set. Problem's test set would be all retargeting method images. Each training-set image was scored (labeled) by an integer score between $[0, 36]$.

Our test images (which is represented by the embedded vectors) were forwarded by the SVR model, and the absolute distance between the ground truth and its actual score is representing the similarity measure of the certain test image.

8.3 parameters

In our test, the following parameters have been observed:

- **C** - discretization points per dimension. In this particular case we have used a fixed C for all dimensions involved, for simplicity. In general, each dimension may have it's own discretization points number
- **G** - embedded groups number. As described in **??** , very long data vectors may consume some serious memory amount through the embedding and learning process, so instead we may use sub-groups of a given dataset. In our case we have sliced-out groups of 4 elements each. Our grouping method have initiated at the 40^{th} element on every SIFT descriptor, and every following group is 4 elements next.
- **k** - rank size in the comparison between users votes and ID similarity results.

SVR model was trained with RBF kernel **rbf** and $C = 1e5$.

8.4 results

Let us observe the results of ID similarity method with the various parameters. As seen in **??** and **??**, ID metrics have outperformed almost all previous metrics in terms of correlation ($-1 \leq \tau \leq 1$) between metrics similarity and users correlation votes.

	Lines/Edges	Faces/People	Texture	Foreground Objects	Geometric Structures	Symmetry	Mean	std	p-value	C	G
BDS	0.040	0.190	0.060	0.167	-0.004	-0.012	0.083	0.268	0.017		
BDW	0.031	0.048	-0.048	0.060	0.004	0.119	0.046	0.181	0.869		
EH	0.043	-0.076	-0.060	-0.079	0.103	0.298	0.004	0.334	0.641		
CL	-0.023	-0.181	-0.071	-0.183	-0.009	0.214	-0.068	0.301	0.384		
RAND	-0.046	-0.014	0.048	-0.032	-0.040	0.143	-0.031	0.284	0.693		
SIFTflow	0.097	0.252	0.119	0.218	0.085	0.071	0.145	0.262	0.031		
EMD	0.220	0.262	0.107	0.226	0.237	0.500	0.251	0.272	1E-5		
ID_1	0.42	0.42	0.40	0.43	0.42	0.41	0.42	0.013		6	5
ID_2	0.45	0.44	**0.44**	0.43	0.41	0.44	0.44	0.014		6	2
ID_3	**0.48**	**0.49**	0.38	**0.46**	**0.45**	**0.47**	**0.45**	0.034		2	2
ID_4	0.37	0.35	0.37	0.41	0.41	0.39	0.38	0.022		2	15

TABLE 8.1: Complete rank correlation, including new ID results ($k = \infty$)

	Lines/Edges	Faces/People	Texture	Foreground Objects	Geometric Structures	Symmetry	Mean	std	p-value	C	G
BDS	0.062	0.280	0.134	0.249	-0.025	-0.247	0.108	0.532	0.005		
BDW	0.213	0.141	0.123	0.115	0.212	0.439	0.200	0.395	0.002		
EH	-0.036	-0.207	-0.331	-0.177	0.111	0.294	-0.071	0.593	0.013		
CL	-0.307	-0.336	-0.433	-0.519	-0.366	0.088	-0.320	0.543	1E-6		
RAND	0.241	0.428	0.312	0.442	0.303	0.002	0.298	0.483	1E-6		
SIFTflow	0.301	0.416	0.216	0.295	0.226	**0.534**	0.326	0.496	1E-6		
EMD	0.220	0.262	0.107	0.226	0.237	0.500	0.251	0.272	1E-5		
ID_1	0.41	0.42	0.41	**0.48**	**0.56**	0.46	0.46	0.051		6	5
ID_2	**0.48**	**0.50**	**0.43**	**0.48**	0.46	0.46	**0.47**	0.022		6	2
ID_3	**0.48**	**0.50**	0.39	0.42	0.38	0.44	0.43	0.045		2	2
ID_4	0.46	0.42	0.35	0.38	0.46	0.39	0.41	0.041		2	15

TABLE 8.2: Rank correlation with respect to the three highest rank results, including new ID ($k = 3$)

This basic fact verifies our correctness of our multidimensional ID embedding method, as well as its sub-sampling grouping method for memory saving. As described in **retargeting** SIFTflow **siftflow** and EMD **EMD** both relies on spatial descriptors, which may be most similar to human eye-brain interpretations of a difference/similarity among pairs of images.

In figures **??**,**??**, we demonstrate ID similarity results on the butterfly image **??**, in comparison to the users votes. As seen on the similarity graph, some retargeting methods such warp **warp** may have not that good similarity result in relation to the reference image, but in this scenario we observed the relative results of the ranking of all methods between users votes and our ID metric.

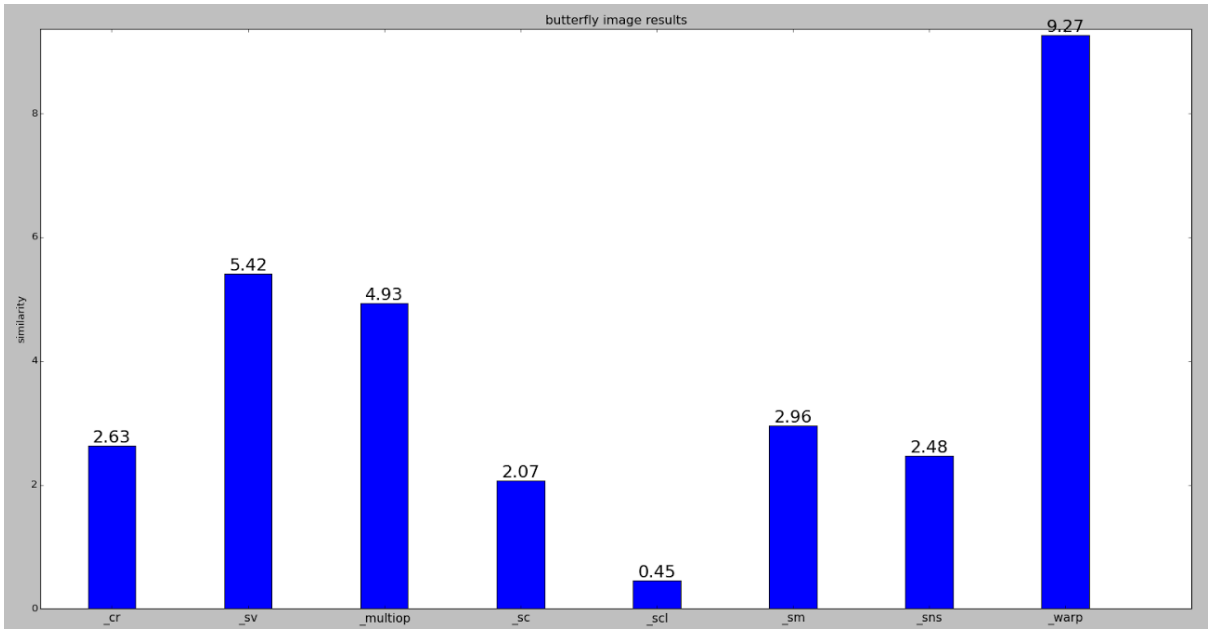

FIGURE 8.3: butterfly similarity results

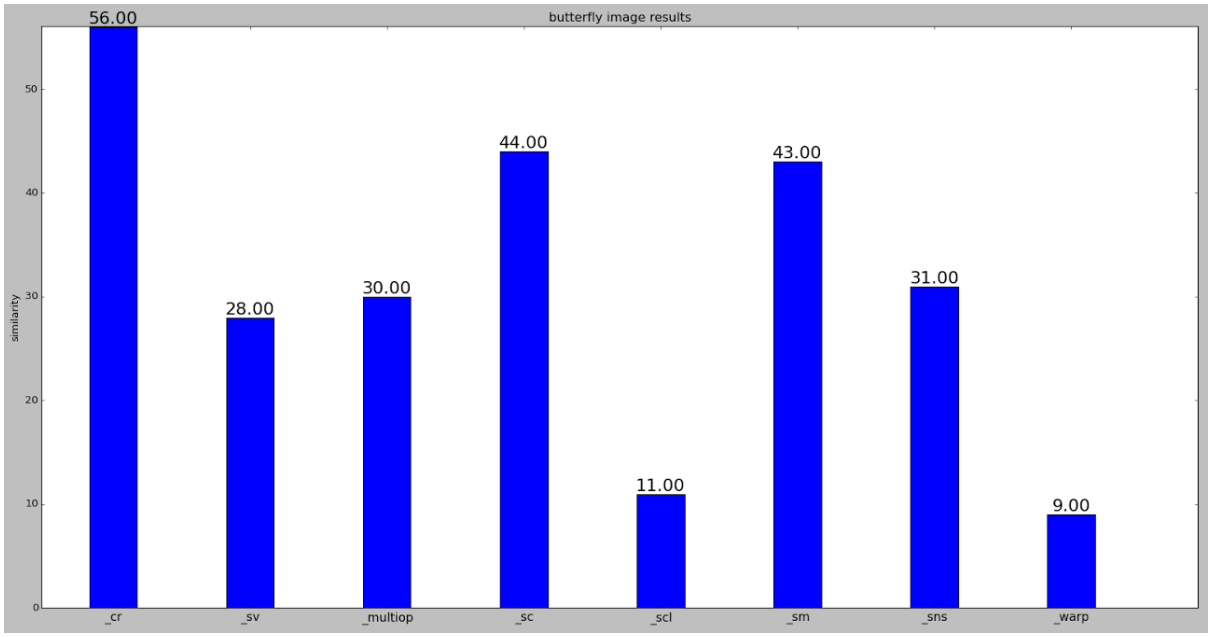

FIGURE 8.4: butterfly similarity users votes

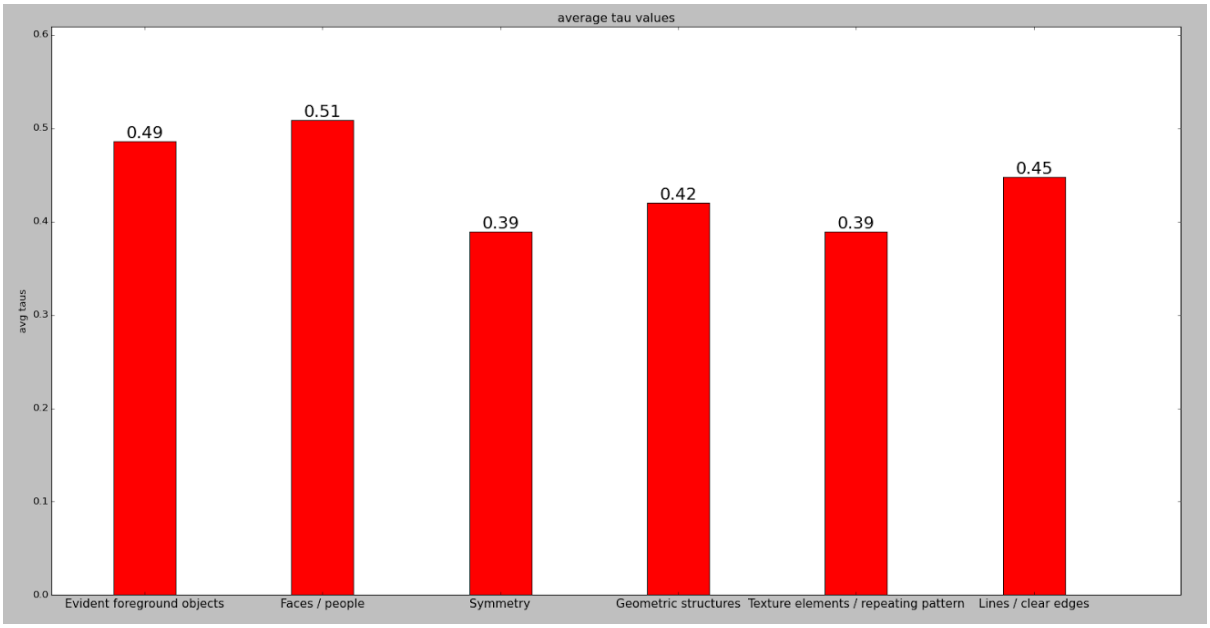

FIGURE 8.5: C = 2 , G = 2 , k = 3

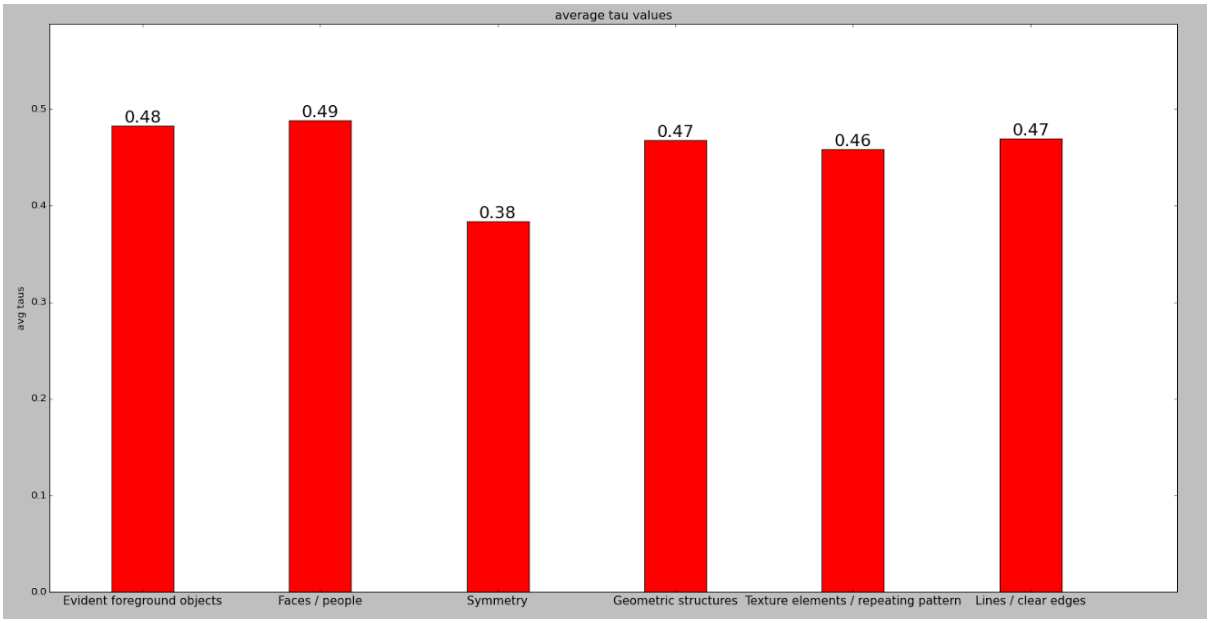

FIGURE 8.6: c2g2kinf

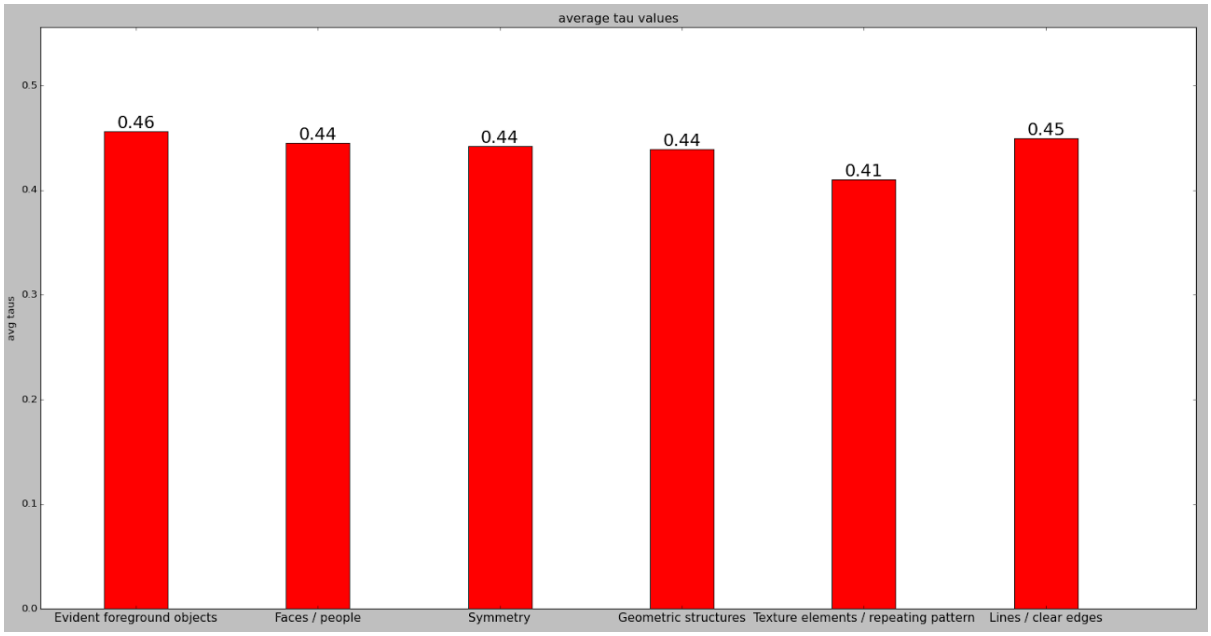

FIGURE 8.7: c6g2kinf

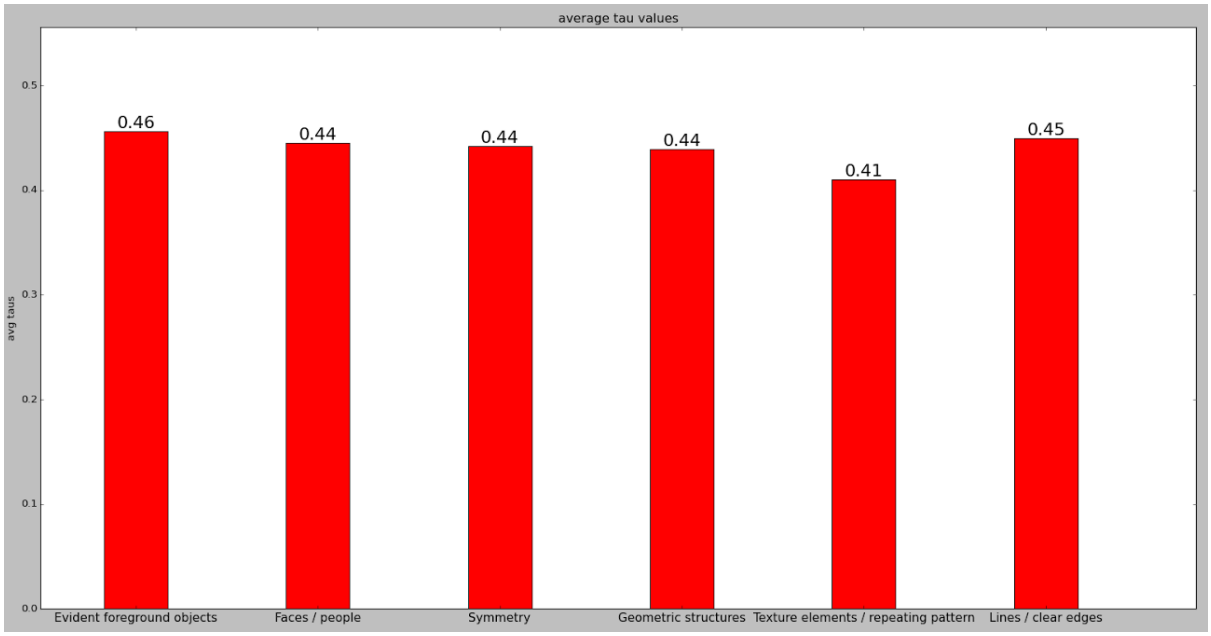

FIGURE 8.8: c6g2kinf

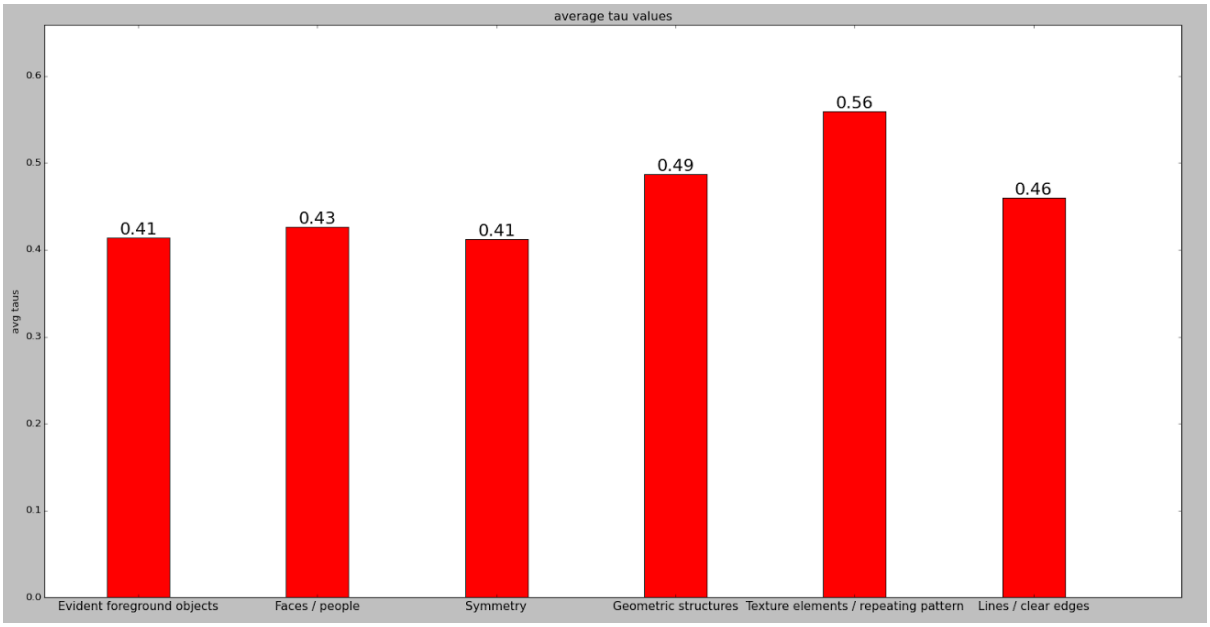

FIGURE 8.9: c6g5k3

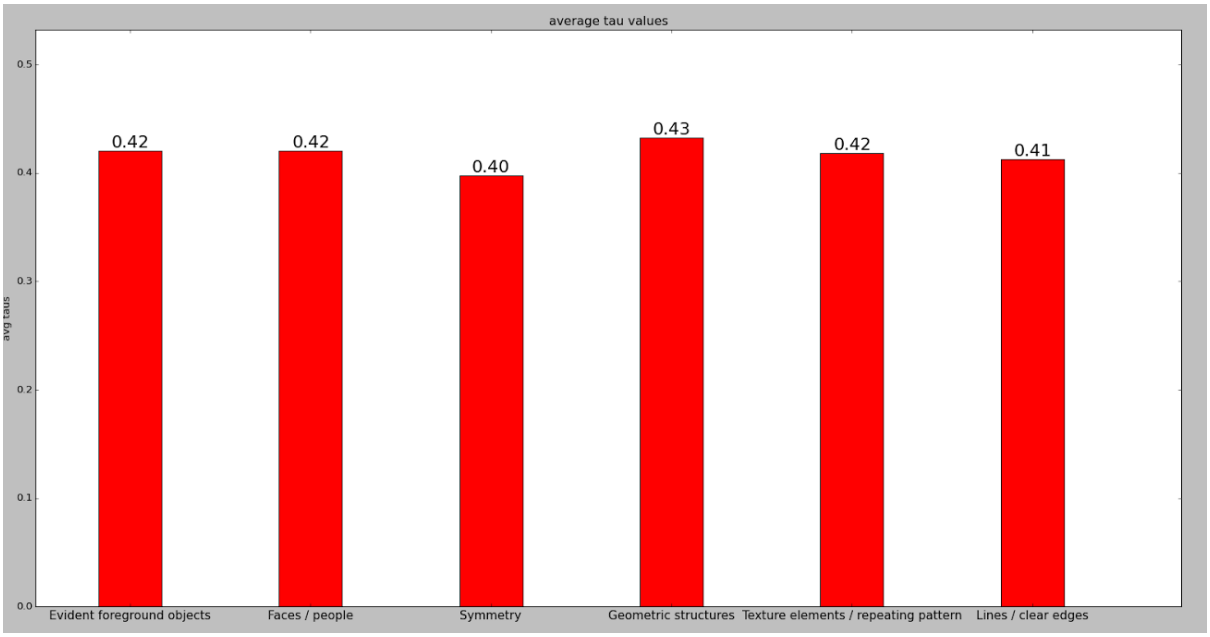

FIGURE 8.10: c6g5kinf

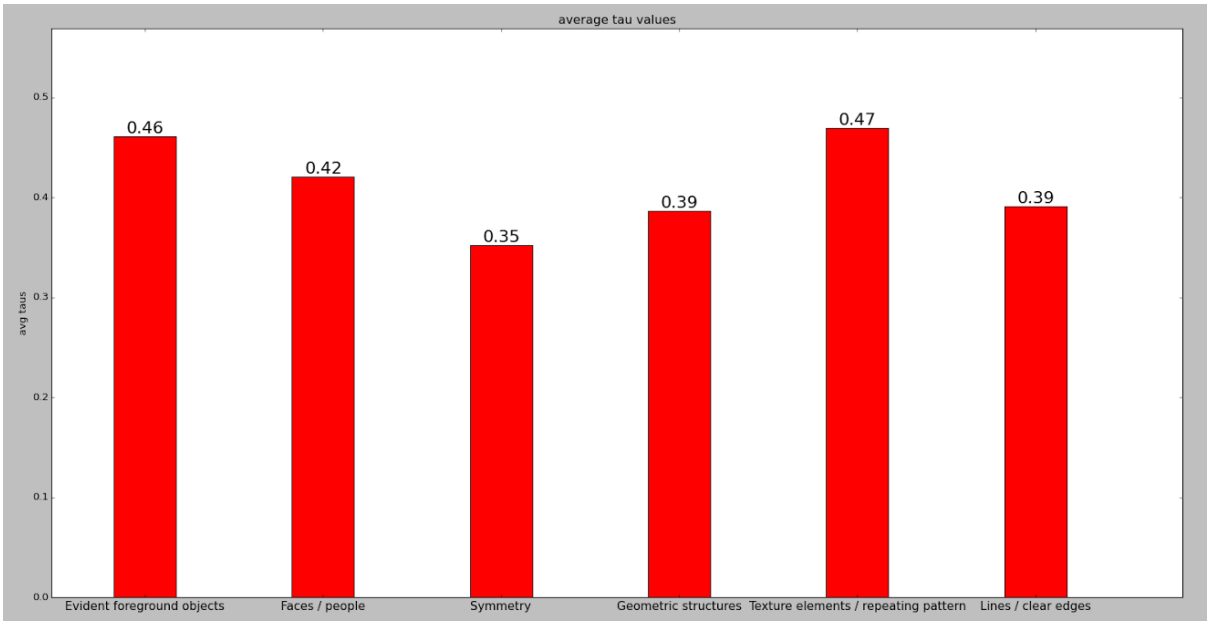

FIGURE 8.11: c2g15k3

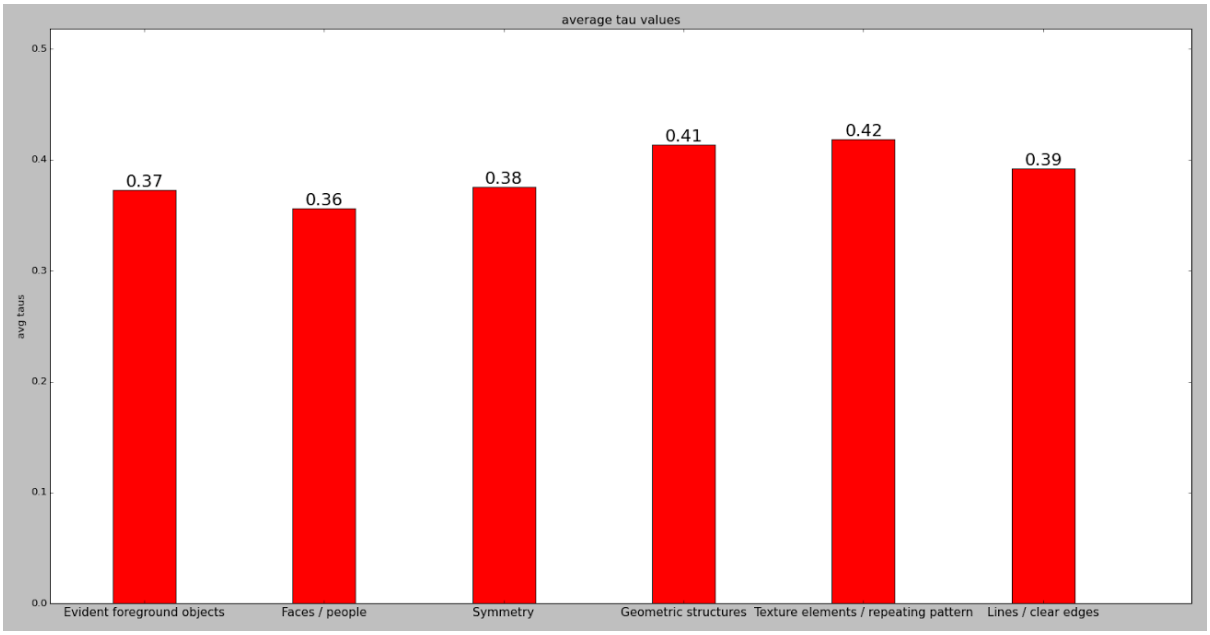

FIGURE 8.12: c2g15kinf

Chapter 9

Conclusions and discussion

Appendix A

Appendix Title Here

Write your Appendix content here. blablabla

Bibliography

[1] Huynen, M. A. and Bork, P. 1998. Measuring genome evolution. *Proceedings of the National Academy of Sciences USA* 95:5849–5856.

[2] Caprara, A. 1997. Sorting by reversals is difficult. In: *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB 97),* New York: ACM. pp. 75-83.

[3] McLysaght, A., Seoighe, C. and Wolfe, K. H. 2000. High frequency of inversions during eukaryote gene order evolution. In Sankoff, D. and Nadeau, J. H., editors, *Comparative Genomics*, Dordrecht, NL: Kluwer Academic Press. pp. 47–58.

[4] Reinelt, G. 1991. *The Traveling Salesman - Computational Solutions for TSP Applications.* Berlin: Springer Verlag.

[5] 5 @articleRubinstein10Comparative, author = Rubinstein, Michael and Gutierrez, Diego and Sorkine, Olga and Shamir, Ariel, title = A Comparative Study of Image Retargeting, journal = ACM Transactions on Graphics (Proc. SIGGRAPH Asia), year = 2010, volume = 29, number = 6, pages = 160:1–160:10,