A Novel Information-Theoretic Point-Set Measure Based on the Jensen-Havrda-Charvat-Tsallis Divergence

Release 0.00

Nicholas J. Tustison, Suyash P. Awate and James C. Gee

October 3, 2008

Penn Image Computing And Science Laboratory
University of Pennsylvania

Abstract

A novel point-set registration algorithm was proposed in [6] based on minimization of the Jensen-Shannon divergence. In this contribution, we generalize this Jensen-Shannon divergence point-set measure framework to the Jensen-Havrda-Charvat-Tsallis divergence. This generalization permits a fine-tuning of the actual divergence measure between robustness and specificity. The principle contribution of this submission is the`itk::JensenHavrdaCharvatTsallisPointSetMetric` class which is derived from the existing `itk::PointSetToPointSetMetric`. In addition, we provide other classes with utility that would extend beyond the point-set measure framework that we provide in this paper. This includes a point-set analogue of the `itk::ImageFunction`, i.e. `itk::PointSetFunction`. From this class we derive the class `itk::ManifoldParzenWindowsPointSetFunction` which provides a Parzen windowing scheme for learning the local structure of point-sets. Finally, we include the `itk::DecomposeTensorFunction` class which wraps the different vnl matrix decomposition schemes for easy use within ITK.

Contents

1 Theoretical Overview

This paper provides a description of recent point-set registration work in which a novel point-set measurement was developed based on the Jensen-Havrda-Charvat-Tsallis (JHCT) entropy family which is a generalization of the Jensen-Shannon (JS) entropy. The basic idea is that these Jensen entropy measures provide a sense of the divergence between multiple probability density functions (PDFs). In fact, it has been shown that the square root of such measures are actually a distance metric with all the associated salient properties [3]. So, given K PDFs we can use standard gradient-based optimization strategies to register these K PDFs.[1] As the discussion centers on registration of point-sets, the question remains how translation from point-sets to PDFs occur. To perform this translation, we choose a Parzen windowing scheme which allows for the local structure to be learned from the point-sets themselves. In the following sections, we first describe how this translation occurs. Then, once, the PDFs have been generated, we describe how the divergence measures and derivatives are calculated. This is followed by a detailed look at the included classes and various examples. Please note that a theoretical paper outlining the details in much greater detail is forthcoming.

1.1 Point-Set to PDF Conversion

Similar to our work, Wang et al. [8] present point-set registration methodology which employs the JS measure. Each point-set is represented as a probability density function through the use of a Gaussian mixture model (GMM) where each point, x_i, specifies a Gaussian center with a constant isotropic covariance. Thus, the k^{th} probability density function, $\mathbf{P}_k(s)$, calculated from the k^{th} point-set (consisting of N_k points) is given by the GMM

$$\mathbf{P}_k(s) = \frac{1}{N_k}\sum_{i=1}^{N_k} G(s; x_i^k, \sigma) \tag{1}$$

where $G(s; x_i^k, \sigma)$ is a normalized Gaussian with mean x_i^k and isotropoic covariance characterized by σ.

However, given that point-sets often represent a sampling of an underlying structure, we modify the conversion process transforming a point-set to its corresponding probability density function to capture that local structure. Whereas previous work used isotropic Gaussians, we use the local point-set neighborhood to estimate an appropriate covariance matrix where the local structure is reflected in the anisotropy of that covariance [7]. For each point, x_i, the associated weighted covariance matrix, $C_{\mathcal{K}_i}$, is given by

$$C_{\mathcal{K}_i} = \frac{\sum_{x_j \in \mathcal{N}_i, x_j \neq x_i} \mathcal{K}(x_i; x_j)(x_i - x_j)^{\mathrm{T}}(x_i - x_j)}{\sum_{x_j \in \mathcal{N}_i, x_j \neq x_i} \mathcal{K}(x_i; x_j)} \tag{2}$$

where $\mathcal{N}_i$ is the local neighborhood of the point x_i and $\mathcal{K}$ is a user-selected neighborhood weighting kernel. We use an isotropic Gaussian for $\mathcal{K}$ as well as a k-d tree structure, i.e. `itk::KdTree`, for efficient determination of $\mathcal{N}_i$.

[1] Please note that our contribution does not include all the components to perform registration. Similar to the image registration solution, one has to decide upon a transformation model and an optimization strategy. We use an earlier contribution from our lab to define a B-spline transformation model [5] and use conjugate gradient descent (CGD) [4] for optimization.

Determination of $C_{\mathcal{K}_i}$ from Equation (2) could potentially result in an ill-conditioned matrix. For this reason, we use the modified covariance, $C_i = C_{\mathcal{K}_i} + \sigma^2 I$ where I is the identity matrix and σ is a user-provided parameter denoting added isotropic Gaussian noise. Thus, the k^{th} probability density function calculated from the k^{th} point-set is given by the GMM

$$\mathbf{P}_k(s) = \frac{1}{N_k} \sum_{i=1}^{N_k} G(s; x_i^k, C_i^k) \tag{3}$$

where $G(s; x_i^k, C_i^k)$ is a normalized Gaussian with mean x_i^k and covariance C_i^k evaluated at s.

1.2 Point-Set Divergence From the Jensen-Havrda-Charvat-Tsallis Entropy

Given K PDFs $\{\mathbf{P}_1, \ldots, \mathbf{P}_K\}$, the JHCT family of divergence measures is given by

$$\mathrm{JHCT}_\alpha(\mathbf{P}_1, \ldots, \mathbf{P}_K) = H_\alpha\left(\sum_{k=1}^{K} \pi_k \mathbf{P}_k\right) - \sum_{k=1}^{K} \pi_k H_\alpha(\mathbf{P}_k) \tag{4}$$

where $H_\alpha(\cdot)$ is the HCT entropy and the set of weights $\{\pi_1, \ldots, \pi_K \,|\pi_k > 0, \sum_{k=1}^{K} \pi_k = 1\}$ determines the relative contribution of the corresponding probability density function to the divergence measure. We assume a weighting scheme of

$$\pi_k = \frac{N_k}{\sum_{i=1}^{K} N_i}. \tag{5}$$

.

Calculation of the derivative of the $JHCT$ divergence with respect to each point is as follows:

$$\frac{\partial JHCT_\alpha}{\partial x_k^i} = \sum_{i=1}^{N_k} \left[\frac{1}{MN} \sum_{k'=1}^{K} \sum_{j=1}^{M_{k'}} \frac{G(s_j^{k'}; x_i^k, C_i^k)(C_i^k)^{-1}(x_i^k - s_j^{k'})}{[\mathbf{P}^*(s_j^{k'})]^{2-\alpha}} - \frac{1}{M_k N} \sum_{j=1}^{M_k} \frac{G(s_j^k; x_i^k, C_i^k)(C_i^k)^{-1}(x_i^k - s_j^k)}{[\mathbf{P}_k(s_j^k)]^{2-\alpha}} \right] \tag{6}$$

where

$$\mathbf{P}^*(s) = \frac{1}{N} \sum_{k=1}^{K} \sum_{i=1}^{N_k} G(s; x_i^k, C_i^k), \quad N = \sum_{k=1}^{K} N_k, \quad M = \sum_{k=1}^{K} M_k. \tag{7}$$

It is well known [1, 2] that $H_\alpha(\cdot)$ reduces to the conventional Shannon entropy as $\alpha \to 1$.

2 Implementation Overview

After keeping the theoretical discussion to a tolerable minimum, we are now in a position to discuss the implementation details of the proposed classes. The classes that we provide in this submission are as follows:

- `itkJensenHavrdaCharvatTsallisPointSetMetric` — this class is derived from the `itk::PointSetToPointSetMetric` and encompasses our principal contribution. Given two input point sets, one "fixed" and one "moving", a PDF is generated from each point-set (see the first section). The user can then calculate the divergence value or the derivative value via the `GetValue()` and the `GetDerivative()` functions, respectively, or both using the `GetValueAndDerivative()` function. The user-specified parameters are as follows:

  - m_UseRegularizationTerm — the divergence in Equation (4) is composed of two terms. In the gradient calculation, the second term acts as a type of regularizer. Therefore, we provide the option of using the first term or both terms.
  - m_UseInputAsSamples — this parameter allows the user to either generate random samples from the generated PDFs or to use the input points as the samples.
  - m_UseAnisotropicCovariances — this parameter allows the user to use isotropic (unlearned) or anisotropic covariances (learned) in generating the PDFs.
  - m_NumberOfFixedSamples — if random samples are to be used, this parameter indicates the number of samples that should be generated from the fixed point-set.
  - m_FixedPointSetSigma — if isotropic covariances are used, this parameter determines σ for each of the fixed point Gaussians. If anisotropic covariances are used, this parameter determines the added isototropic Gaussian noise.
  - m_FixedKernelSigma — if isotropic covariances are used, this parameter is irrelevant. If anisotropic covariances are used, this parameter determines the Gaussian kernel used in Equation (2).
  - m_FixedCovarianceKNeighborhood — this parameter specifies the number of neighbors to be used in constructing the anisotropic covariances. If isotropic covariances are used, this parameter is irrelevant.
  - m_FixedEvaluationKNeighborhood — this parameter specifies the number of neighbors used in evaluating the PDF at a specific point.
  - m_NumberOfMovingSamples — same as the fixed point set.
  - m_MovingPointSetSigma — same as for the fixed point set.
  - m_MovingKernelSigma — same as for the fixed point set.
  - m_MovingEvaluationKNeighborhood — same as for the fixed point set.
  - m_MovingCovarianceKNeighborhood — same as for the fixed point set.
  - m_Alpha — this clamped parameter which is allowed to vary between $[1,2]$ provides a trade-off between specificity and robustness.
  - m_UseWithRespectToTheMovingPointSet — it is our intent to employ this metric for symmetric point-set registration so the derivative is needed for both directions. Since generation of the kd-tree for each point-set takes $O(n \log n)$ time, we simply allow the user to specify which derivative is desired so that both can be obtained without recreating the kd-trees.

- itkPointSetFunction — this base class is analogous to the itk::itkImageFunction class in that given a point-set, each of the derived classes is meant to return a value.

- itkManifoldParzenWindowsPointSetFunction — this class is derived from the itkPointSetFunction. It encapsulates both anisotropic and isotropic GMMs. This class is used by the itkJensenHavrdaCharvatTsallisPointSetMetric class. Many of the parameters used in this class are the same as those in the itkJensenHavrdaCharvatTsallisPointSetMetric class.

- itkGaussianProbabilityDensityFunction — this class is an extensive modification of the itk::GaussianDensityFunction class which allows for generation of random samples.

- itkDecomposeTensorFunction — this class provides an easy interface to various vnl matrix decomposition routines. These functions are:

- EvaluateEigenDecomposition
- EvaluateSymmetricEigenDecomposition
- EvaluateQRDecomposition
- EvaluateSVDDecomposition
- EvaluateSVDEcoonomyDecomposition
- EvaluateCholeskyDecomposition
- EvaluateDeterminant

Although this class has more generic utility, in the specific case discussed in this submission, it is used in the itkGaussianProbabilityDensityFunction class to perform an eigen-decomposition of the covariance matrix for generating random samples.

3 Sample Usage

3.1 itkDecomposeTensorFunction

The itk::DecomposeTensorFunction class is templated over the InputMatrixType, a RealType (defaults to float), and an OutputMatrixType (defaults to a itk::VariableSizeMatrix type). Usage is straightforward as the following code snippet illustrates for performing eigen-decomposition. First, we create an arbitrary 3×3 matrix.

```
InputMatrixType M( 3, 3 );
M(0, 0) = 1;
M(0, 1) = 2;
M(0, 2) = 3;
M(1, 0) = 2;
M(1, 1) = 5;
M(1, 2) = 4;
M(2, 0) = 3;
M(2, 1) = 4;
M(2, 2) = 9;
```

We then call the desired function (eigen-decomposition in the example below).

```
/**
 * Eigen-Decomposition
 */
OutputMatrixType D, V;
try
  {
  decomposer->EvaluateEigenDecomposition( M, D, V );
  }
catch(...)
  {
  std::cerr << "EvaluateEigenDecomposition:  Exception thrown."
            << std::endl;
  return EXIT_FAILURE;
  }
```

D and V hold the eigenvalues and eigenvectors, respectively.

3.2 itkJensenHavrdaCharvatTsallisPointSetMetric

The `itk::JensenHavrdaCharvatTsallisPointSetMetric` class is templated over the point-set type. The user also needs to know that unlike the flexibility of the `itk::PointSetToPointSetMetric` class, the only acceptable transform type is the `itk::IdentityTransform`, due to the use of the kd-tree for facilitating evaluation. Usage is described with the following example. We first read in the two point-sets. Please note that the point-set reader class `itkLabeledPointSetFileReader` was developed by the authors to read in labeled point-sets from various file formats (such as labeled images and vtk files) but it is not intended to be included in ITK.

```
const unsigned int Dimension = 2;

typedef float RealType;
typedef itk::PointSet<RealType, Dimension> PointSetType;

typedef itk::LabeledPointSetFileReader<PointSetType> ReaderType;

ReaderType::Pointer fixedPointSetReader = ReaderType::New();
fixedPointSetReader->SetFileName( argv[1] );
fixedPointSetReader->Update();

ReaderType::Pointer movingPointSetReader = ReaderType::New();
movingPointSetReader->SetFileName( argv[2] );
movingPointSetReader->Update();
```

After instantiation of the metric class, we set the fixed and moving point-sets as well as the other parameters.

```
typedef itk::JensenHavrdaCharvatTsallisPointSetMetric<PointSetType>
  PointSetMetricType;
PointSetMetricType::Pointer pointSetMetric = PointSetMetricType::New();
pointSetMetric->SetMovingPointSet( movingPointSetReader->GetOutput() );
pointSetMetric->SetMovingPointSetSigma( atof( argv[4] ) );
pointSetMetric->SetMovingEvaluationKNeighborhood( 30 );
pointSetMetric->SetFixedPointSet( fixedPointSetReader->GetOutput() );
pointSetMetric->SetFixedPointSetSigma( atof( argv[3] ) );
pointSetMetric->SetFixedEvaluationKNeighborhood( 30 );
pointSetMetric->SetUseInputAsSamples( atoi( argv[7] ) );
pointSetMetric->SetUseAnisotropicCovariances( atoi( argv[6] ) );
pointSetMetric->SetAlpha( atof( argv[5] ) );

if( pointSetMetric->GetUseAnisotropicCovariances() )
  {
  pointSetMetric->SetFixedCovarianceKNeighborhood( 5 );
  pointSetMetric->SetFixedKernelSigma( 2 *
    pointSetMetric->GetFixedPointSetSigma() );
  pointSetMetric->SetMovingCovarianceKNeighborhood( 5 );
  pointSetMetric->SetMovingKernelSigma( 2 *
    pointSetMetric->GetMovingPointSetSigma() );
  }

if( !pointSetMetric->GetUseInputAsSamples() )
  {
  pointSetMetric->SetNumberOfFixedSamples( 1000 );
  pointSetMetric->SetNumberOfMovingSamples( 1250 );
  }
```

We then initialize the point-set metric and calculate the value and derivatives with respect to the fixed point-set

```
  pointSetMetric->Initialize();

  PointSetMetricType::DefaultTransformType::ParametersType parameters;
  parameters.Fill( 0 );

  pointSetMetric->SetUseWithRespectToTheMovingPointSet( false );
  PointSetMetricType::DerivativeType gradientFixed;
  pointSetMetric->GetDerivative( parameters, gradientFixed );

  std::cout << "Fixed gradient: " << std::endl;
  std::cout << gradientFixed << std::endl;

  PointSetMetricType::MeasureType measureFixed
    = pointSetMetric->GetValue( parameters );

  std::cout << "Fixed value: " << std::endl;
  std::cout << measureFixed << std::endl << std::endl;

  PointSetMetricType::MeasureType measureFixedTest;
  PointSetMetricType::DerivativeType gradientFixedTest;

  pointSetMetric->GetValueAndDerivative( parameters,
    measureFixedTest, gradientFixedTest );
```

and the moving point-set.

```
  pointSetMetric->SetUseWithRespectToTheMovingPointSet( true );
  PointSetMetricType::DerivativeType gradientMoving;
  pointSetMetric->GetDerivative( parameters, gradientMoving );

  std::cout << "Moving gradient: " << std::endl;
  std::cout << gradientMoving << std::endl;

  PointSetMetricType::MeasureType measureMoving
    = pointSetMetric->GetValue( parameters );

  std::cout << "Moving value: " << std::endl;
  std::cout << measureMoving << std::endl;

  PointSetMetricType::MeasureType measureMovingTest;
  PointSetMetricType::DerivativeType gradientMovingTest;

  pointSetMetric->GetValueAndDerivative( parameters,
    measureMovingTest, gradientMovingTest );
```

Using ITK-SNAP we drew the two face outlines shown in Figure 1. These point-sets were then used as input for different permutations of different parameters for the tests given in `CMakeLists.txt`. The resulting derivatives for the test `JHCT_3` are illustrated in Figure 2.

References

[1] J. Burbea and C. R. Rao. On the convexity of some divergence measures on entropy functions. *IEEE Transactions on Information Theory*, 28:489–495, 1982. 1.2

[2] M. Gell-Mann and C. Tsallis. *Nonextensive Entropy*. Oxford University Press, 2004. 1.2

[3] A. Majtey, P. Lamberti, and A. Plastino. A monoparametric family of metrics for statistical mechanics. *Physica A*, 344:547–553, 2004. 1

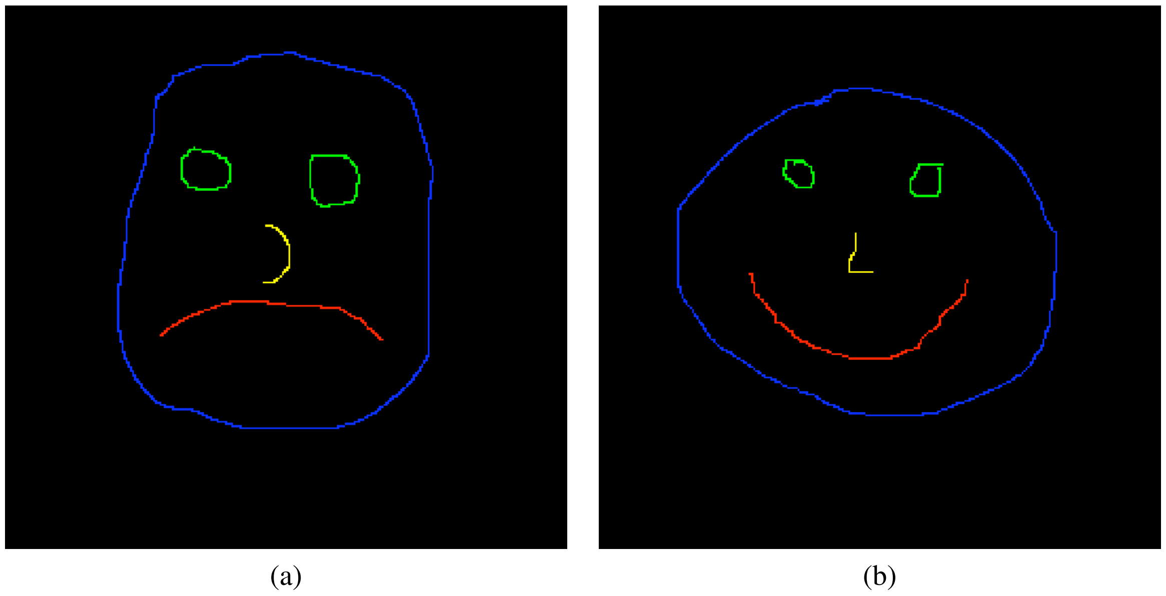

Figure 1: (a) “Frowney-faced” point-set and (b) “Smiley-faced” point-set used to demonstrate the point-set divergence measure.

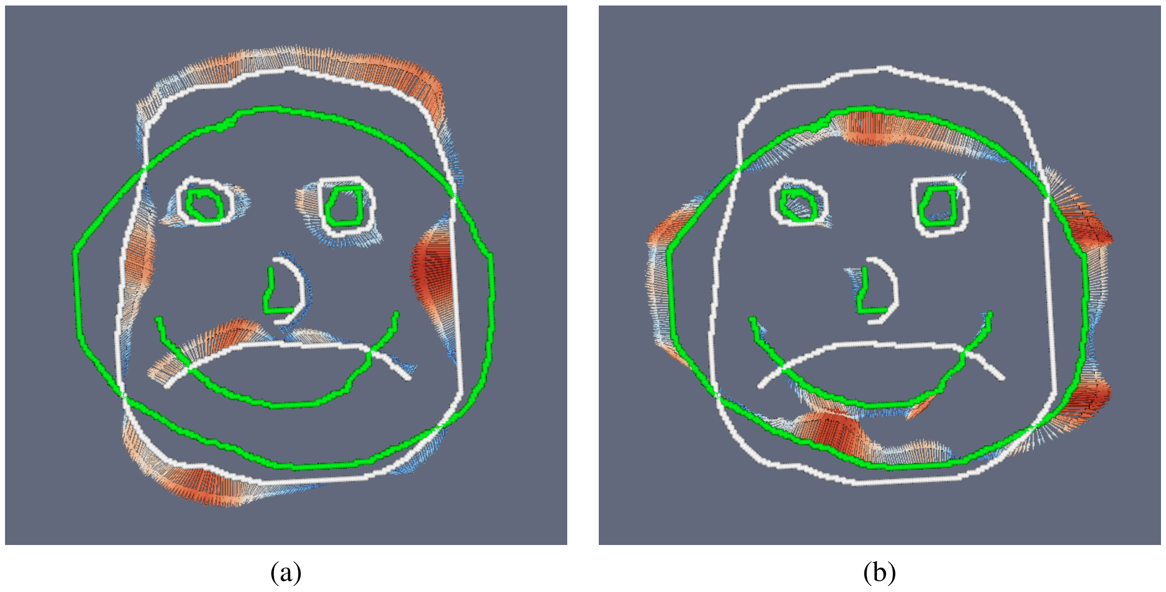

Figure 2: “Frowney-faced” (white) and “smiley-faced” (green) point-set displayed in Paraview along with the gradient with respect to (a) the fixed point-set and (b) the moving point-set.

[4] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2 edition, 1992. 1

[5] N. J. Tustison and J. C. Gee. N-d C^k B-spline scattered data approximation. *The Insight Journal*, 2005. 1

[6] Nicholas J. Tustison, Jing Cai, Talissa A. Altes, G. Wilson Miller, Eduard E. de Lange, John P. Mugler III, and James C. Gee. Pulmonary kinematics from 3-d hyperpolarized helium-3 tagged magnetic resonance imaging. In *Proceedings of the International Society for Magnetic Resonance in Medicine*, 2008. submitted. (document)

[7] Pascal Vincent and Yoshua Bengio. Manifold parzen windows. In S. Thrun, S. Becker, and K. Obermayer, editors, *Advances in Neural Information Prcessing Systems*, pages 825–832. MIT Press, 2003. 1.1

[8] Fei Wang, Baba C. Vemuri, and Anand Rangarajan. Groupwise point pattern registration using a novel CDF-based Jensen-Shannon divergence. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006. 1.1