Sliding wavelet coefficients

We use a feature representation that incorporates the use of a density estimation using wavelets, which brings us to a d-dimensional hypersphere. However, the original density estimation was over the 2- or 3-dimensional space, i.e. each coefficient of represents the amplitude of the wavelet in a region of 2-d or 3-d space. We then compute distances between the probability density functions through use of the angle or arc between the points on the hypersphere; intuitively, this distance metric measures how different the corresponding regions are in amplitude.

However, if the shapes are not aligned (specifically, if there are non-rigid transformations between them), there may be mistakes in using this distance metric. The goal of sliding the wavelet coefficients, as described in Shape L'Ane Rouge[**?**], is to reduce non-rigid differences between the densities before taking the distance between them.

Review of wavelet density estimation

Our goal is to compute a robust distance metric between shapes such that the essential information inherent in the shape is captured while the irrelevant information is reduced. This follows the paradigm where similarities between shapes are considered after removing certain classes of transformations[**?**, **?**].

We first take the problem of representing shapes in such a way. Given a 2D or 3D point cloud, we attempt to measure it either directly as a probability density function or after applying certain transformations (e.g. the Laplace-Beltrami operator). We estimate $\sqrt{p(x)}$ and then obtain $p(x) = (\sqrt{p(x)})^2$ in order to sidestep many of the problems associated with estimating probability density functions.

Let $\boldsymbol{x} \in \mathbb{R}^2$, j_1 be some stopping scale level for multiscale decomposition, and $\boldsymbol{k} \in \mathbb{Z}^2$ be an index into the spatial location of the basis. In 2D, the wavelet expansion of densities is given as follows:

$$\sqrt{p(x)} = \sum_{j_0,\boldsymbol{k}} \alpha_{j_0,\boldsymbol{k}} \phi_{j_0,\boldsymbol{k}}(\boldsymbol{x}) + \sum_{j \geq j_0,\boldsymbol{k}}^{j_1} \sum_{w=1}^{3} \beta_{j,\boldsymbol{k}}^{w} \psi_{j,\boldsymbol{k}}^{w}(\boldsymbol{x}) \tag{1}$$

with the coefficients $\alpha_{j_0,\boldsymbol{k}}$ and $\beta_{j,\boldsymbol{k}}^{w}$ representing "how dense" the probability density function is at that spatial point.

Because of the unit integrability of probability density functions, we can interpret a coefficient as a point on the unit hypersphere, and can use the corresponding distance metric (i.e. angle or arc length between any two points).

Motivation for sliding

Suppose there are two shapes which are identical except for a translation such that none of the points overlapped. If we take the above density estimation

technique, the wavelet coefficients will have a high distance on the unit hypersphere (i.e. the dot product between the two vectors is 0, so we get a distance of $\frac{\pi}{2}$). However, these shapes are the same and our goal is to have a low distance between similar shapes. In other words, we want to reduce the effect of these transformations that don't inform us about the intrinsic difference between shapes. If we had a method to match up corresponding points with each other and then take the hypersphere distance, we could have a much more robust distance metric to use.

In Shape L'Ane Rouge, we attempt to warp one density into the shape of another before taking the hypersphere distance between them, using linear assignment.

Use of linear assignment

Linear assignment informally attempts to match up a set of objects with another set of objects (usually with the same cardinality) in such a way as to minimze cost. We use this to match up each coefficient in the vector with another in order to find the maximum overlap between the two. We penalize large shifts by using some multiple λ of the Euclidean distance between regions in the original point/point-cloud representation (otherwise, any shape would be able to fit any other shape).

Implementation

Let $\boldsymbol{c_1}, \boldsymbol{c_2}$ be the two coefficient vectors that we are taking the distance between. We attempt to match $\boldsymbol{c_1}$ with $\boldsymbol{c_2}$ using linear assignment, with a cost matrix

$$C = -(\boldsymbol{c_1} \otimes \boldsymbol{c_2}) + \lambda D$$

where $\boldsymbol{c_1} \otimes \boldsymbol{c_2}$ is the outer product between the two coefficient vectors, D is distance matrix with d_{ij} = the Euclidean distance between the ith and jth regions that the coefficients correspond to, and λ is some weight value that determines how important distance is to the cost.

The distance matrix is constructed by reconstructing the 2- or 3-dimensional grid under which the coefficients were formed using MATLAB's `ndgrid` function, finding the points `pts`, and then computing the pairwise distance using MATLAB's `squareform(pdist(pts))`.