Introduction to Geometric Learning in Python with Geomstats

Nina Miolane, Nicolas Guigui, Hadi Zaatiti, Christian Shewmake, Hatem Hajri, Daniel Brooks, Alice Le Brigant, Johan Mathe, Benjamin Hou, Yann Thanwerdas, et al.

▶ To cite this version:

Nina Miolane, Nicolas Guigui, Hadi Zaatiti, Christian Shewmake, Hatem Hajri, et al.. Introduction to Geometric Learning in Python with Geomstats. SciPy 2020 - 19th Python in Science Conference, Jul 2020, Austin, Texas, United States. pp.48-57, 10.25080/Majora-342d178e-007 . hal-02908006

HAL Id: hal-02908006

https://hal.inria.fr/hal-02908006

Submitted on 28 Jul 2020

Introduction to Geometric Learning in Python with Geomstats

Nina Miolane[‡][*], Nicolas Guigui[§], Hadi Zaatiti, Christian Shewmake, Hatem Hajri, Daniel Brooks, Alice Le Brigant, Johan Mathe, Benjamin Hou, Yann Thanwerdas, Stefan Heyder, Olivier Peltre, Niklas Koep, Yann Cabanes, Thomas Gerald, Paul Chauchat, Bernhard Kainz, Claire Donnat, Susan Holmes, Xavier Pennec

https://youtu.be/Ju-Wsd84uG0

Abstract—There is a growing interest in leveraging differential geometry in the machine learning community. Yet, the adoption of the associated geometric computations has been inhibited by the lack of a reference implementation. Such an implementation should typically allow its users: (i) to get intuition on concepts from differential geometry through a hands-on approach, often not provided by traditional textbooks; and (ii) to run geometric machine learning algorithms seamlessly, without delving into the mathematical details. To address this gap, we present the open-source Python package `geomstats` and introduce hands-on tutorials for differential geometry and geometric machine learning algorithms - Geometric Learning - that rely on it. Code and documentation: `github.com/geomstats/geomstats` and `geomstats.ai`.

Index Terms—differential geometry, statistics, manifold, machine learning

Introduction

Data on manifolds arise naturally in different fields. Hyperspheres model directional data in molecular and protein biology [KH05] and some aspects of 3D shapes [JDM12], [HVS+16]. Density estimation on hyperbolic spaces arises to model electrical impedances [HKKM10], networks [AS14], or reflection coefficients extracted from a radar signal [CBA15]. Symmetric Positive Definite (SPD) matrices are used to characterize data from Diffusion Tensor Imaging (DTI) [PFA06], [YZLM12] and functional Magnetic Resonance Imaging (fMRI) [STK05]. These manifolds are curved, differentiable generalizations of vector spaces. Learning from data on manifolds thus requires techniques from the mathematical discipline of differential geometry. As a result, there is a growing interest in leveraging differential geometry in the machine learning community, supported by the fields of Geometric Learning and Geometric Deep Learning [BBL+17].

Despite this need, the adoption of differential geometric computations has been inhibited by the lack of a reference implementation. Projects implementing code for geometric tools are often custom-built for specific problems and are not easily reused. Some Python packages do exist, but they mainly focus on optimization (Pymanopt [TKW16], Geoopt [BG18], [Koc19], McTorch [MJK+18]), are dedicated to a single manifold (PyRiemann [Bar15], PyQuaternion [Wyn14], PyGeometry [Cen12]), or lack unit-tests and continuous integration (TheanoGeometry [KS17]). An open-source, low-level implementation of differential geometry and associated learning algorithms for manifold-valued data is thus thoroughly welcome.

`Geomstats` is an open-source Python package built for machine learning with data on non-linear manifolds [MGLB+]: a field called Geometric Learning. The library provides object-oriented and extensively unit-tested implementations of essential manifolds, operations, and learning methods with support for different execution backends - namely NumPy, PyTorch, and TensorFlow. This paper illustrates the use of `geomstats` through hands-on introductory tutorials of Geometric Learning. These tutorials enable users: (i) to build intuition for differential geometry through a hands-on approach, often not provided by traditional textbooks; and (ii) to run geometric machine learning algorithms seamlessly without delving into the lower-level computational or mathematical details. We emphasize that the tutorials are not meant to replace theoretical expositions of differential geometry and geometric learning [Pos01], [PSF19]. Rather, they will complement them with an intuitive, didactic, and engineering-oriented approach.

Presentation of Geomstats

The package geomstats is organized into two main modules: geometry and learning. The module `geometry` implements low-level differential geometry with an object-oriented paradigm and two main parent classes: `Manifold` and `RiemannianMetric`. Standard manifolds like the `Hypersphere` or the `Hyperbolic` space are classes that inherit from `Manifold`. At the time of writing, there are over 15 manifolds implemented in `geomstats`. The class `RiemannianMetric` provides computations related to Riemannian geometry on such manifolds such as the inner product of two tangent vectors at a base point, the geodesic distance between two points, the Exponential and Logarithm maps at a base point, and many others.

The module `learning` implements statistics and machine learning algorithms for data on manifolds. The code is object-oriented and classes inherit from `scikit-learn` base classes and mixins such as `BaseEstimator`, `ClassifierMixin`, or `RegressorMixin`. This module provides implementations

* Corresponding author: nmiolane@stanford.edu
‡ Stanford University
§ Université Côte d'Azur, Inria

of Fréchet mean estimators, K-means, and principal component analysis (PCA) designed for manifold data. The algorithms can be applied seamlessly to the different manifolds implemented in the library.

The code follows international standards for readability and ease of collaboration, is vectorized for batch computations, undergoes unit-testing with continuous integration, and incorporates both TensorFlow and PyTorch backends to allow for GPU acceleration. The package comes with a visualization module that enables users to visualize and further develop an intuition for differential geometry. In addition, the datasets module provides instructive toy datasets on manifolds. The repositories examples and notebooks provide convenient starting points to get familiar with `geomstats`.

First Steps

To begin, we need to install `geomstats`. We follow the installation procedure described in the first steps of the online documentation. Next, in the command line, we choose the backend of interest: NumPy, PyTorch or TensorFlow. Then, we open the iPython notebook and import the backend together with the visualization module. In the command line:

```
export GEOMSTATS_BACKEND=numpy
```

then, in the notebook:

```
import geomstats.backend as gs
import geomstats.visualization as visualization

visualization.tutorial_matplotlib()
```

```
INFO: Using numpy backend
```

Modules related to `matplotlib` and `logging` should be imported during setup too. More details on setup can be found on the documentation website: `geomstats.ai`. All standard NumPy functions should be called using the `gs.` prefix - e.g. `gs.exp`, `gs.log` - in order to automatically use the backend of interest.

Tutorial: Statistics and Geometric Statistics

This tutorial illustrates how Geometric Statistics and Learning differ from traditional Statistics. Statistical theory is usually defined for data belonging to vector spaces, which are linear spaces. For example, we know how to compute the mean of a set of numbers or of multidimensional arrays.

Now consider a non-linear space: a manifold. A manifold M of dimension m is a space that is possibly curved but that looks like an m-dimensional vector space in a small neighborhood of every point. A sphere, like the earth, is a good example of a manifold. What happens when we apply statistical theory defined for linear vector spaces to data that does not naturally belong to a linear space? For example, what happens if we want to perform statistics on the coordinates of world cities lying on the earth's surface: a sphere? Let us compute the mean of two data points on the sphere using the traditional definition of the mean.

```
from geomstats.geometry.hypersphere import \
    Hypersphere

n_samples = 2
sphere = Hypersphere(dim=2)
points_in_manifold = sphere.random_uniform(
     n_samples=n_samples)
```

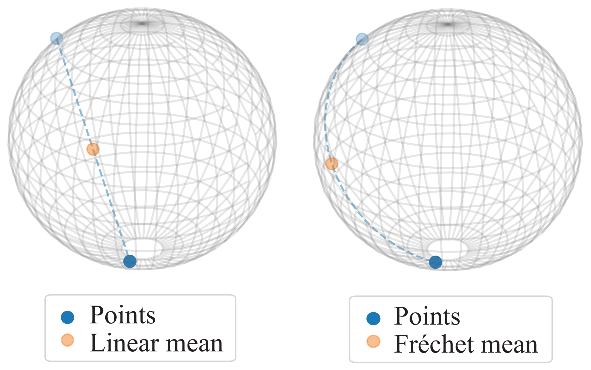


Fig. 1: *Left: Linear mean of two points on the sphere. Right: Fréchet mean of two points on the sphere. The linear mean does not belong to the sphere, while the Fréchet mean does. This illustrates how linear statistics can be generalized to data on manifolds, such as points on the sphere.*

```
linear_mean = gs.sum(
    points_in_manifold, axis=0) / n_samples
```

The result is shown in Figure 1 (left). What happened? The mean of two points on a manifold (the sphere) is not on the manifold. In our example, the mean of these cities is not on the earth's surface. This leads to errors in statistical computations. The line `sphere.belongs(linear_mean)` returns `False`. For this reason, researchers aim to build a theory of statistics that is - by construction - compatible with any structure with which we equip the manifold. This theory is called Geometric Statistics, and the associated learning algorithms: Geometric Learning.

In this specific example of mean computation, Geometric Statistics provides a generalization of the definition of "mean" to manifolds: the Fréchet mean.

```
from geomstats.learning.frechet_mean import \
    FrechetMean

estimator = FrechetMean(metric=sphere.metric)
estimator.fit(points_in_manifold)
frechet_mean = estimator.estimate_
```

Notice in this code snippet that `geomstats` provides classes and methods whose API will be instantly familiar to users of the widely-adopted `scikit-learn`. We plot the result in Figure 1 (right). Observe that the Fréchet mean now belongs to the surface of the sphere!

Beyond the computation of the mean, `geomstats` provides statistics and learning algorithms on manifolds that leverage their specific geometric structure. Such algorithms rely on elementary operations that are introduced in the next tutorial.

Tutorial: Elementary Operations for Data on Manifolds

The previous tutorial showed why we need to generalize traditional statistics for data on manifolds. This tutorial shows how to perform the elementary operations that allow us to "translate" learning algorithms from linear spaces to manifolds.

We import data that lie on a manifold: the world cities dataset, that contains coordinates of cities on the earth's surface. We visualize it in Figure 2.

```
import geomstats.datasets.utils as data_utils

data, names = data_utils.load_cities()
```

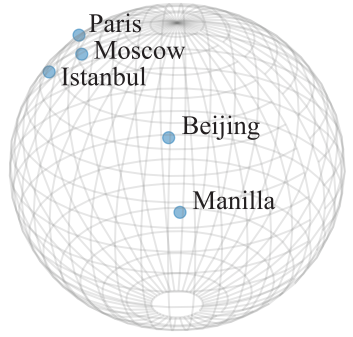


Fig. 2: *Subset of the world cities dataset, available in* `geomstats` *with the function* `load_cities` *from the module* `datasets.utils`*. Cities' coordinates are data on the sphere, which is an example of a manifold.*

How can we compute with data that lie on such a manifold? The elementary operations on a vector space are addition and subtraction. In a vector space (in fact seen as an affine space), we can add a vector to a point and subtract two points to get a vector. Can we generalize these operations in order to compute on manifolds?

For points on a manifold, such as the sphere, the same operations are not permitted. Indeed, adding a vector to a point will not give a point that belongs to the manifold: in Figure 3, adding the black tangent vector to the blue point gives a point that is outside the surface of the sphere. So, we need to generalize to manifolds the operations of addition and subtraction.

On manifolds, the exponential map is the operation that generalizes the addition of a vector to a point. The exponential map takes the following inputs: a point and a tangent vector to the manifold at that point. These are shown in Figure 3 using the blue point and its tangent vector, respectively. The exponential map returns the point on the manifold that is reached by "shooting" with the tangent vector from the point. "Shooting" means following a "geodesic" on the manifold, which is the dotted path in Figure 3. A geodesic, roughly, is the analog of a straight line for general manifolds - the path whose, length, or energy, is minimal between two points, where the notions of length and energy are defined by the Riemannian metric. This code snippet shows how to compute the exponential map and the geodesic with `geomstats`.

```python
from geomstats.geometry.hypersphere import \
    Hypersphere

sphere = Hypersphere(dim=2)

initial_point = paris = data[19]
vector = gs.array([1, 0, 0.8])
tangent_vector = sphere.to_tangent(
     vector, base_point=initial_point)

end_point = sphere.metric.exp(
    tangent_vector, base_point=initial_point)

geodesic = sphere.metric.geodesic(
    initial_point=initial_point,
    initial_tangent_vec=tangent_vector)
```

Similarly, on manifolds, the logarithm map is the operation that generalizes the subtraction of two points on vector spaces. The logarithm map takes two points on the manifold as inputs and returns the tangent vector required to "shoot" from one point to

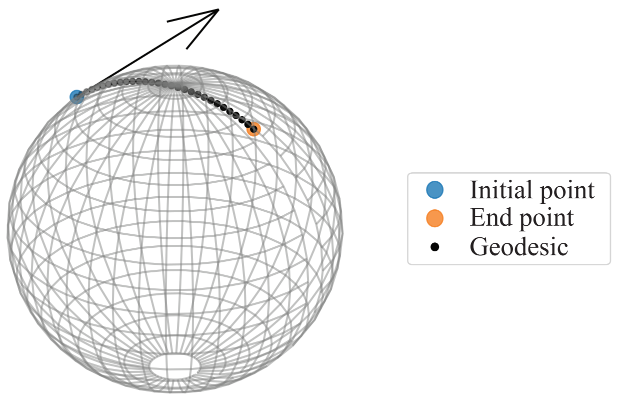


Fig. 3: *Elementary operations on manifolds illustrated on the sphere. The exponential map at the initial point (blue point) shoots the black tangent vector along the geodesic, and gives the end point (orange point). Conversely, the logarithm map at the initial point (blue point) takes the end point (orange point) as input, and outputs the black tangent vector. The geodesic between the blue point and the orange point represents the path of shortest length between the two points.*

the other. At any point, it is the inverse of the exponential map. In Figure 3, the logarithm of the orange point at the blue point returns the tangent vector in black. This code snippet shows how to compute the logarithm map with `geomstats`.

```python
log = sphere.metric.log(
    point=end_point, base_point=initial_point)
```

We emphasize that the exponential and logarithm maps depend on the "Riemannian metric" chosen for a given manifold: observe in the code snippets that they are not methods of the `sphere` object, but rather of its `metric` attribute. The Riemannian metric defines the notion of exponential, logarithm, geodesic and distance between points on the manifold. We could have chosen a different metric on the sphere that would have changed the distance between the points: with a different metric, the "sphere" could, for example, look like an ellipsoid.

Using the exponential and logarithm maps instead of linear addition and subtraction, many learning algorithms can be generalized to manifolds. We illustrated the use of the exponential and logarithm maps on the sphere only; yet, `geomstats` provides their implementation for over 15 different manifolds in its `geometry` module with support for a variety of Riemannian metrics. Consequently, `geomstats` also implements learning algorithms on manifolds, taking into account their specific geometric structure by relying on the operations we just introduced. The next tutorials show more involved examples of such geometric learning algorithms.

Tutorial: Classification of SPD Matrices

Tutorial context and description

We demonstrate that any standard machine learning algorithm can be applied to data on manifolds while respecting their geometry. In the previous tutorials, we saw that linear operations (mean, linear weighting, addition and subtraction) are not defined on manifolds. However, each point on a manifold has an associated tangent space which is a vector space. As such, in the tangent space, these operations are well defined! Therefore, we can use the logarithm map (see Figure 3 from the previous tutorial) to go from points on manifolds to vectors in the tangent space at a reference point. This first strategy enables the use of traditional learning algorithms on manifolds.

A second strategy can be designed for learning algorithms, such as K-Nearest Neighbors classification, that rely only on distances or dissimilarity metrics. In this case, we can compute the pairwise distances between the data points on the manifold, using the method `metric.dist`, and feed them to the chosen algorithm.

Both strategies can be applied to any manifold-valued data. In this tutorial, we consider symmetric positive definite (SPD) matrices from brain connectomics data and perform logistic regression and K-Nearest Neighbors classification.

SPD matrices in the literature

Before diving into the tutorial, let us recall a few applications of SPD matrices in the machine learning literature. SPD matrices are ubiquitous across many fields [CS16], either as input of or output to a given problem. In DTI for instance, voxels are represented by "diffusion tensors" which are 3x3 SPD matrices representing ellipsoids in their structure. These ellipsoids spatially characterize the diffusion of water molecules in various tissues. Each DTI thus consists of a field of SPD matrices, where each point in space corresponds to an SPD matrix. These matrices then serve as inputs to regression models. In [YZLM12] for example, the authors use an intrinsic local polynomial regression to compare fiber tracts between HIV subjects and a control group. Similarly, in fMRI, it is possible to extract connectivity graphs from time series of patients' resting-state images [WZD+13]. The regularized graph Laplacians of these graphs form a dataset of SPD matrices. This provides a compact summary of brain connectivity patterns which is useful for assessing neurological responses to a variety of stimuli, such as drugs or patient's activities.

More generally speaking, covariance matrices are also SPD matrices which appear in many settings. Covariance clustering can be used for various applications such as sound compression in acoustic models of automatic speech recognition (ASR) systems [SMA10] or for material classification [FHP15], among others. Covariance descriptors are also popular image or video descriptors [HHLS16].

Lastly, SPD matrices have found applications in deep learning. The authors of [GWB+19] show that an aggregation of learned deep convolutional features into an SPD matrix creates a robust representation of images which outperforms state-of-the-art methods for visual classification.

Manifold of SPD matrices

Let us recall the mathematical definition of the manifold of SPD matrices. The manifold of SPD matrices in n dimensions is embedded in the General Linear group of invertible matrices and defined as:

$$SPD = \left\{ S \in \mathbb{R}_{n\times n} : S^T = S, \forall z \in \mathbb{R}^n, z \neq 0, z^T S z > 0 \right\}.$$

The class `SPDMatricesSpace` inherits from the class `EmbeddedManifold` and has an `embedding_manifold` attribute which stores an object of the class `GeneralLinear`. SPD matrices in 2 dimensions can be visualized as ellipses with principal axes given by the eigenvectors of the SPD matrix, and the length of each axis proportional to the square-root of the corresponding eigenvalue. This is implemented in the

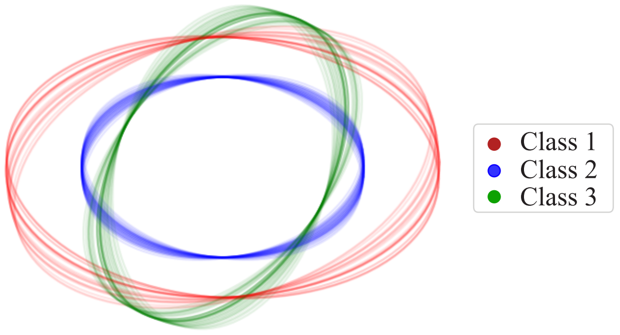


Fig. 4: Simulated dataset of SPD matrices in 2 dimensions. We observe 3 classes of SPD matrices, illustrated with the colors red, green, and blue. The centroid of each class is represented by an ellipse of larger width.

`visualization` module of `geomstats`. We generate a toy data-set and plot it in Figure 4 with the following code snippet.

```
import geomstats.datasets.sample_sdp_2d as sampler

n_samples = 100
dataset_generator = sampler.DatasetSPD2D(
    n_samples, n_features=2, n_classes=3)

ellipsis = visualization.Ellipsis2D()
for i,x in enumerate(data):
    y = sampler.get_label_at_index(i, labels)
    ellipsis.draw(
        x, color=ellipsis.colors[y], alpha=.1)
```

Figure 4 shows a dataset of SPD matrices in 2 dimensions organized into 3 classes. This visualization helps in developing an intuition on the connectomes dataset that is used in the upcoming tutorial, where we will classify SPD matrices in 28 dimensions into 2 classes.

Classifying brain connectomes in Geomstats

We now delve into the tutorial in order to illustrate the use of traditional learning algorithms on the tangent spaces of manifolds implemented in `geomstats`. We use brain connectome data from the MSLP 2014 Schizophrenia Challenge. The connectomes are correlation matrices extracted from the time-series of resting-state fMRIs of 86 patients at 28 brain regions of interest: they are points on the manifold of SPD matrices in $n = 28$ dimensions. Our goal is to use the connectomes to classify patients into two classes: schizophrenic and control. First we load the connectomes and display two of them as heatmaps in Figure 5.

```
import geomstats.datasets.utils as data_utils

data, patient_ids, labels = \
    data_utils.load_connectomes()
```

Multiple metrics can be used to compute on the manifold of SPD matrices [DKZ09]. As mentionned in the previous tutorial, different metrics define different geodesics, exponential and logarithm maps and therefore different algorithms on a given manifold. Here, we import two of the most commonly used metrics on the SPD matrices, the log-Euclidean metric and the affine-invariant metric [PFA06], but we highlight that `geomstats` contains many more. We also check that our connectome data indeed belongs to the manifold of SPD matrices:

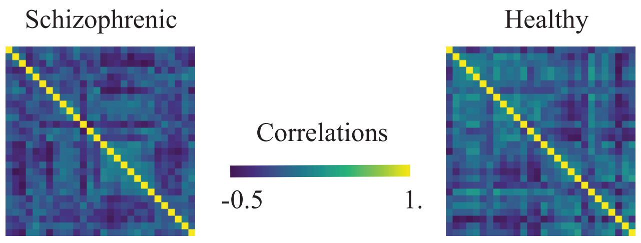


Fig. 5: Subset of the connectomes dataset, available in `geomstats` *with the function* `load_connectomes` *from the module* `datasets.utils`*. Connectomes are correlation matrices of 28 time-series extracted from fMRI data: they are elements of the manifold of SPD matrices in 28 dimensions. Left: connectome of a schizophrenic subject. Right: connectome of a healthy control.*

```
import geomstats.geometry.spd_matrices as spd

manifold = spd.SPDMatrices(n=28)
le_metric = spd.SPDMetricLogEuclidean(n=28)
ai_metric = spd.SPDMetricAffine(n=28)
logging.info(gs.all(manifold.belongs(data)))
```

```
INFO: True
```

Great! Now, although the sum of two SPD matrices is an SPD matrix, their difference or their linear combination with non-positive weights are not necessarily. Therefore we need to work in a tangent space of the SPD manifold to perform simple machine learning that relies on linear operations. The `preprocessing` module with its `ToTangentSpace` class allows to do exactly this.

```
from geomstats.learning.preprocessing import \
    ToTangentSpace
```

`ToTangentSpace` has a simple purpose: it computes the Fréchet Mean of the data set, and takes the logarithm map of each data point from the mean. This results in a data set of tangent vectors at the mean. In the case of the SPD manifold, these are simply symmetric matrices. `ToTangentSpace` then squeezes each symmetric matrix into a 1d-vector of size `dim = 28 * (28 + 1) / 2`, and outputs an array of shape `[n_connectomes, dim]`, which can be fed to your favorite `scikit-learn` algorithm.

We emphasize that `ToTangentSpace` computes the mean of the input data, and thus should be used in a pipeline (as e.g. `scikit-learn`'s `StandardScaler`) to avoid leaking information from the test set at train time.

```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_validate

pipeline = make_pipeline(
    ToTangentSpace(le_metric), LogisticRegression(C=2))
```

We use a logistic regression on the tangent space at the Fréchet mean to classify connectomes, and evaluate the model with cross-validation. With the log-Euclidean metric we obtain:

```
result = cross_validate(pipeline, data, labels)
logging.info(result['test_score'].mean())
```

```
INFO: 0.67
```

And with the affine-invariant metric, replacing `le_metric` by `ai_metric` in the above snippet:

```
INFO: 0.71
```

We observe that the result depends on the metric. The Riemannian metric indeed defines the notion of the logarithm map, which is used to compute the Fréchet Mean and the tangent vectors corresponding to the input data points. Thus, changing the metric changes the result. Furthermore, some metrics may be more suitable than others for different applications. Indeed, we find published results that show how useful geometry can be with data on the SPD manifold (e.g [WAZF18], [NDV+14]).

We saw how to use the representation of points on the manifold as tangent vectors at a reference point to fit any machine learning algorithm, and we compared the effect of different metrics on the manifold of SPD matrices. Another class of machine learning algorithms can be used very easily on manifolds with `geomstats`: those relying on dissimilarity matrices. We can compute the matrix of pairwise Riemannian distances, using the *dist* method of the Riemannian metric object. In the following code-snippet, we use `ai_metric.dist` and pass the corresponding matrix `pairwise_dist` of pairwise distances to `scikit-learn`'s *K*-Nearest-Neighbors (KNN) classification algorithm:

```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(
    metric='precomputed')

result = cross_validate(
    classifier, pairwise_dist, labels)
logging.info(result['test_score'].mean())
```

```
INFO: 0.72
```

This tutorial showed how to leverage `geomstats` to use standard learning algorithms for data on a manifold. In the next tutorial, we see a more complicated situation: the data points are not provided by default as elements of a manifold. We will need to use the low-level `geomstats` operations to design a method that embeds the dataset in the manifold of interest. Only then, we can use a learning algorithm.

Tutorial: Learning Graph Representations with Hyperbolic Spaces

Tutorial context and description

This tutorial demonstrates how to make use of the low-level geometric operations in `geomstats` to implement a method that embeds graph data into the hyperbolic space. Thanks to the discovery of hyperbolic embeddings, learning on Graph-Structured Data (GSD) has seen major achievements in recent years. It had been speculated for years that hyperbolic spaces may better represent GSD than Euclidean spaces [Gro87] [KPK+10] [BPK10] [ASM13]. These speculations have recently been shown effective through concrete studies and applications [NK17] [CCD17] [SDSGR18] [GZH+19]. As outlined by [NK17], Euclidean embeddings require large dimensions to capture certain complex relations such as the Wordnet noun hierarchy. On the other hand, this complexity can be captured by a lower-dimensional model of hyperbolic geometry such as the hyperbolic space of two dimensions [SDSGR18], also called the hyperbolic plane. Additionally, hyperbolic embeddings provide better visualizations of clusters on graphs than their Euclidean counterparts [CCD17].

This tutorial illustrates how to learn hyperbolic embeddings in `geomstats`. Specifically, we will embed the Karate Club graph dataset, representing the social interactions of the members of a university Karate club, into the Poincaré ball. Note that we will omit implementation details but an unabridged example and detailed notebook can be found on GitHub in the `examples` and `notebooks` directories of `geomstats`.

Hyperbolic spaces and machine learning applications

Before going into this tutorial, we review a few applications of hyperbolic spaces in the machine learning literature. First, Hyperbolic spaces arise in information and learning theory. Indeed, the space of univariate Gaussians endowed with the Fisher metric densities is a hyperbolic space [CSS05]. This characterization is used in various fields, for example in image processing, where each image pixel can be represented by a Gaussian distribution [AVF14], or in radar signal processing where the corresponding echo is represented by a stationary Gaussian process [ABY13]. Hyperbolic spaces can also be seen as continuous versions of trees and are therefore interesting when learning representations of hierarchical data [NK17]. Hyperbolic Geometric Graphs (HGG) have also been suggested as a promising model for social networks - where the hyperbolicity appears through a competition between similarity and popularity of an individual [PKS+12] and in learning communities on large graphs [GZH+19].

Hyperbolic space

Let us recall the mathematical definition of the hyperbolic space. The n-dimensional hyperbolic space H_n is defined by its embedding in the $(n+1)$-dimensional Minkowski space as:

$$H_n = \left\{x \in \mathbb{R}^{n+1} : -x_1^2 + \ldots + x_{n+1}^2 = -1\right\}. \qquad (1)$$

In `geomstats`, the hyperbolic space is implemented in the class `Hyperboloid` and `PoincareBall`, which use different coordinate systems to represent points. These classes inherit from the class `EmbeddedManifold` and have an `embedding_manifold` attribute which stores an object of the class `Minkowski`. The 2-dimensional hyperbolic space is called the hyperbolic plane or Poincaré disk.

Learning graph representations with hyperbolic spaces in `geomstats`

Parameters and Initialization: We now proceed with the tutorial embedding the Karate club graph in a hyperbolic space. In the Karate club graph, each node represents a member of the club, and each edge represents an undirected relation between two members. We first load the Karate club dataset, display it in Figure 6 and print information regarding its nodes and vertices to provide insights into the graph's complexity.

```
karate_graph = data_utils.load_karate_graph()
nb_vertices_by_edges = (
    [len(e_2) for _, e_2 in
        karate_graph.edges.items()])
logging.info(
    'Number of vertices: %s', len(karate_graph.edges))
logging.info(
    'Mean edge-vertex ratio: %s',
    (sum(nb_vertices_by_edges, 0) /
        len(karate_graph.edges)))
```

```
INFO: Number of vertices: 34
INFO: Mean edge-vertex ratio: 4.588235294117647
```

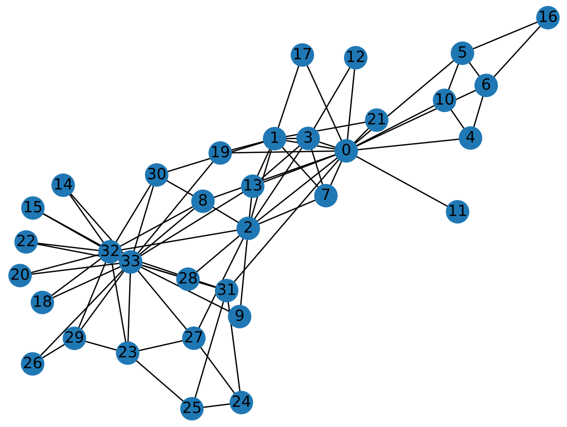


Fig. 6: *Karate club dataset, available in `geomstats` with the function `load_karate_graph` from the module `datasets.utils`. This dataset is a graph, where each node represents a member of the club and each edge represents a tie between two members of the club.*

Parameter	Description	Value
dim	Dimension of the hyperbolic space	2
max_epochs	Number of embedding iterations	15
lr	Learning rate	0.05
n_negative	Number of negative samples	2
context_size	Size of the context for each node	1
karate_graph	Instance of the `Graph` class returned by the function `load_karate_graph` in `datasets.utils`	

TABLE 1: *Hyperparameters used to embed the Karate Club Graph into a hyperbolic space.*

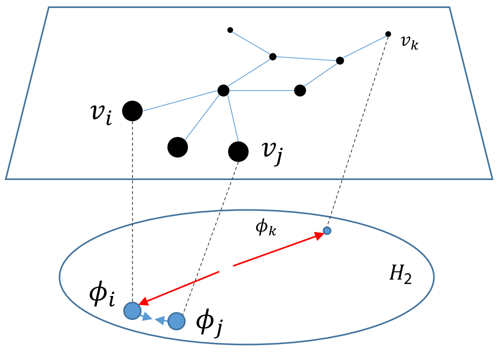
Table 1 defines the parameters needed to embed this graph into a hyperbolic space. The number of hyperbolic dimensions should be high ($n > 10$) only for graph datasets with a large number of nodes and edges. In this tutorial we consider a dataset with only 34 nodes, which are the 34 members of the Karate club. The Poincaré ball of two dimensions is therefore sufficient to capture the complexity of the graph. We instantiate an object of the class `PoincareBall` in `geomstats`.

```
from geomstats.geometry.poincare_ball
    import PoincareBall

hyperbolic_manifold = PoincareBall(dim=2)
```

Other parameters such as `max_epochs` and `lr` will be tuned specifically for each dataset, either manually leveraging visualization functions or through a grid/random search that looks for parameter values maximizing some performance function (a measure for cluster separability, normalized mutual information (NMI), or others). Similarly, the number of negative samples and context size are hyperparameters and will be further discussed below.

Learning the embedding by optimizing a loss function: Denote V as the set of nodes and $E \subset V \times V$ the set of edges of the graph. The goal of hyperbolic embedding is to provide a faithful and exploitable representation of the graph. This goal is mainly achieved by preserving first-order proximity that encourages nodes sharing edges to be close to each other. We can additionally pre-



→ Gradient direction for context samples
→ Gradient direction for negative samples

Fig. 7: Embedding of the graph's nodes $\{v_i\}_i$ as points $\{\phi_i\}_i$ of the hyperbolic plane H_2, also called the Poincaré ball of 2 dimensions. The blue and red arrows represent the direction of the gradient of the loss function $\mathscr{L}$ from Equation 2. This brings context samples closer and separates negative samples.

serve second-order proximity by encouraging two nodes sharing the "same context", i.e. not necessarily directly connected but sharing a neighbor, to be close. We define a context size (here equal to 1) and call two nodes "context samples" if they share a neighbor, and "negative samples" otherwise. To preserve first and second-order proximities, we adopt the following loss function similar to [NK17] and consider the "negative sampling" approach from [MSC+13]:

$$\mathscr{L} = -\sum_{v_i \in V} \sum_{v_j \in C_i} \left[\log(\sigma(-d^2(\phi_i, \phi'_j))) + \sum_{v_k \sim \mathscr{P}_n} \log(\sigma(d^2(\phi_i, \phi'_k))) \right] \tag{2}$$

where $\sigma(x) = (1+e^{-x})^{-1}$ is the sigmoid function and $\phi_i \in H_2$ is the embedding of the i-th node of V, C_i the nodes in the context of the i-th node, $\phi'_j \in H_2$ the embedding of $v_j \in C_i$. Negatively sampled nodes v_k are chosen according to the distribution $\mathscr{P}_n$ such that $\mathscr{P}_n(v) = (\deg(v)^{3/4}).(\sum_{v_i \in V} \deg(v_i)^{3/4})^{-1}$.

Intuitively one can see in Figure 7 that minimizing $\mathscr{L}$ makes the distance between ϕ_i and ϕ_j smaller, and the distance between ϕ_i and ϕ_k larger. Therefore by minimizing $\mathscr{L}$, one obtains representative embeddings.

Riemannian optimization: Following the literature on optimization on manifolds [GBH18], we use the following gradient updates to optimize $\mathscr{L}$:

$$\phi^{t+1} = \mathrm{Exp}_{\phi^t}\left(-lr \frac{\partial \mathscr{L}}{\partial \phi}\right)$$

where ϕ is a parameter of $\mathscr{L}$, $t \in \{1, 2, \cdots\}$ is the iteration number, and lr is the learning rate. The formula consists of first computing the usual gradient of the loss function for the direction in which the parameter should move. The Riemannian exponential map Exp is the operation introduced in the second tutorial: it takes a base point ϕ^t and a tangent vector T and returns the point ϕ^{t+1}. The Riemannian exponential map is a method of the `PoincareBallMetric` class in the `geometry` module of `geomstats`. It allows us to implement a straightforward generalization of standard gradient update in the Euclidean case. To compute the gradient of $\mathscr{L}$, we need to compute the gradients of: (i) the squared distance $d^2(x,y)$ on the hyperbolic space, (ii) the log sigmoid $\log(\sigma(x))$, and (iii) the composition of (i) with (ii).

For (i), we use the formula proposed by [ABY13] which uses the Riemannian logarithmic map. Like the exponential Exp, the logarithmic map is implemented under the `PoincareBallMetric`.

```python
def grad_squared_distance(point_a, point_b, manifold):
    log = manifold.metric.log(point_b, point_a)
    return -2 * log
```

For (ii), we compute the well-known gradient of the logarithm of the sigmoid function as: $(\log \sigma)'(x) = (1+\exp(x))^{-1}$. For (iii), we apply the composition rule to obtain the gradient of $\mathscr{L}$. The following function computes $\mathscr{L}$ and its gradient on the context samples, while ignoring the part dealing with the negative samples for simplicity of exposition. The code implementing the whole `loss` function is available on GitHub.

```python
def loss(example, context_embedding, manifold):

    context_distance = manifold.metric.squared_dist(
        example, context_embedding)
    context_loss = log_sigmoid(-context_distance)
    context_log_sigmoid_grad = -grad_log_sigmoid(
        -context_distance)

    context_distance_grad = grad_squared_distance(
        example, context_embedding, manifold)

    context_grad = (context_log_sigmoid_grad
        * context_distance_grad)

    return context_loss, -context_grad
```

Capturing the graph structure: We perform initialization computations that capture the graph structure. We compute random walks initialized from each v_i up to some length (five by default). The context nodes v_j will be later picked from the random walk of v_i.

```python
random_walks = karate_graph.random_walk()
```

Negatively sampled nodes v_k are chosen according to the previously defined probability distribution function $\mathscr{P}_n(v_k)$ implemented as

```python
negative_table_parameter = 5
negative_sampling_table = []

for i, nb_v in enumerate(nb_vertices_by_edges):
    negative_sampling_table += (
        [i] * int((nb_v**(3. / 4.)))
          * negative_table_parameter)
```

Numerically optimizing the loss function: We can now embed the Karate club graph into the Poincaré disk. The details of the initialization are provided on GitHub. The array `embeddings` contains the embeddings ϕ_i's of the nodes `v_i`'s of the current iteration. At each iteration, we compute the gradient of $\mathscr{L}$. The graph nodes are then moved in the direction pointed by the gradient. The movement of the nodes is performed by following geodesics in the Poincaré disk in the gradient direction. In practice, the key to obtaining a representative embedding is to carefully tune the learning rate so that all of the nodes make small movements at each iteration.

A first level loop iterates over the epochs while the table `total_loss` records the value of $\mathscr{L}$ at each iteration. A second level nested loop iterates over each path in the previously computed random walks. Observing these walks, note that nodes having many edges appear more often. Such nodes can be considered as important crossroads and will therefore be subject to a greater number of embedding updates. This is one of the main reasons why random walks have proven to be effective in capturing the structure of graphs. The context of each v_i will be the set of nodes v_j belonging to the random walk from v_i. The `context_size` specified earlier will limit the length of the walk to be considered. Similarly, we use the same `context_size` to limit the number of negative samples. We find ϕ_i from the `embeddings` array.

A third and fourth level nested loops will iterate on each v_j and v_k. From within, we find ϕ'_j and ϕ'_k and call the `loss` function to compute the gradient. Then the Riemannian exponential map is applied to find the new value of ϕ_i as we mentioned before.

```
for epoch in range(max_epochs):
    total_loss = []
    for path in random_walks:
        for example_index,
                one_path in enumerate(path):
            context_index = path[max(
                0, example_index - context_size):
                min(example_index + context_size,
                len(path))]
            negative_index = gs.random.randint(
                negative_sampling_table.shape[0],
                size=(len(context_index), n_negative))
            negative_index = (
                negative_sampling_table[negative_index])
            example_embedding = embeddings[one_path]
            for one_context_i, one_negative_i in \
                zip(context_index, negative_index):
                context_embedding = (
                    embeddings[one_context_i])
                negative_embedding = (
                    embeddings[one_negative_i])
                l, g_ex = loss(
                    example_embedding,
                    context_embedding,
                    negative_embedding,
                    hyperbolic_manifold)
                total_loss.append(l)

                example_to_update = (
                    embeddings[one_path])
                embeddings[one_path] = (
                    hyperbolic_metric.exp(
                    -lr * g_ex, example_to_update))
    logging.info(
        'iteration %d loss_value %f',
        epoch, sum(total_loss, 0) / len(total_loss))
```

```
INFO: iteration 0 loss_value 1.819844
INFO: iteration 14 loss_value 1.363593
```

Figure 8 shows the graph embedding at different iterations with the true labels of each node represented with color. Notice how the embedding at convergence separates well the two clusters. Thus, it seems that we have found a useful representation of the graph.

To demonstrate the usefulness of the embedding learned, we show how to apply a K-means algorithm in the hyperbolic plane to predict the label of each node in an unsupervised approach. We use the `learning` module of `geomstats` and instantiate an object of the class `RiemannianKMeans`. Observe again how `geomstats` classes follow `scikit-learn`'s API. We set the number of clusters and plot the results.

```
from geomstats.learning.kmeans import RiemannianKMeans

kmeans = RiemannianKMeans(
```

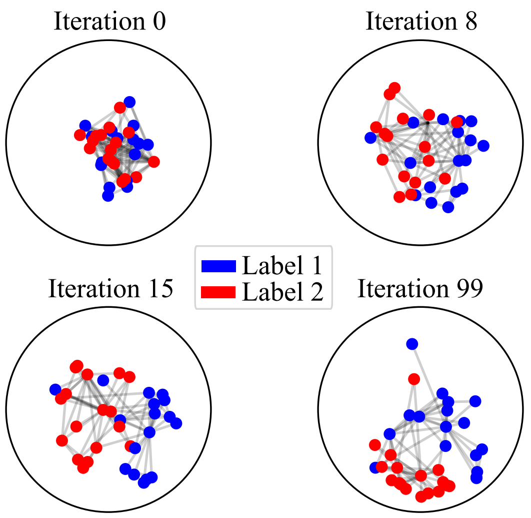


Fig. 8: Embedding of the Karate club graph into the hyperbolic plane at different iterations. The colors represent the true label of each node.

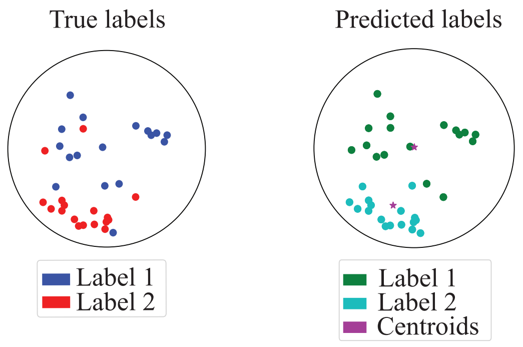


Fig. 9: Results of the Riemannian K-means algorithm on the Karate graph dataset embedded in the hyperbolic plane. Left: True labels associated to the club members. Right: Predicted labels via Riemannian K-means on the hyperbolic plane. The centroids of the clusters are shown with a star marker.

```
    hyperbolic_manifold.metric, n_clusters=2,
    mean_method='frechet-poincare-ball')
centroids = kmeans.fit(X=embeddings, max_iter=100)
labels = kmeans.predict(X=embeddings)
```

Figure 9 shows the true labels versus the predicted ones: the two groups of the karate club members have been well separated!

Conclusion

This paper demonstrates the use of `geomstats` in performing geometric learning on data belonging to manifolds. These tutorials, as well as many other learning examples on a variety of manifolds, can be found at `geomstats.ai`. We hope that this hands-on presentation of Geometric Learning will help to further democratize the use of differential geometry in the machine learning community.

Acknowledgements

This work is partially supported by the National Science Foundation, grant NSF DMS RTG 1501767, the Inria-Stanford associated team GeomStats, and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement G-Statistics No. 786854).

References

[ABY13] Marc Arnaudon, Frédéric Barbaresco, and Le Yang. Riemannian medians and means with applications to radar signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 7(4):595–604, 2013. URL: https://ieeexplore.ieee.org/document/6514112, doi:10.1109/JSTSP.2013.2261798.

[AS14] Dena Asta and Cosma Rohilla Shalizi. Geometric Network Comparison. *Journal of Machine Learning Research*, 2014. URL: http://arxiv.org/abs/1411.1350, doi:10.1109/PES.2006.1709566.

[ASM13] Aaron B Adcock, Blair D Sullivan, and Michael W Mahoney. Tree-like structure in large social and information networks. In *2013 IEEE 13th International Conference on Data Mining*, pages 1–10. IEEE, 2013. URL: https://ieeexplore.ieee.org/document/6729484, doi:10.1109/ICDM.2013.77.

[AVF14] Jesus Angulo and Santiago Velasco-Forero. Morphological processing of univariate Gaussian distribution-valued images based on Poincaré upper-half plane representation. In Frank Nielsen, editor, *Geometric Theory of Information*, Signals and Communication Technology, pages 331–366. Springer International Publishing, 5 2014. URL: https://hal.archives-ouvertes.fr/hal-00795012, doi:10.1007/978-3-319-05317-2_12.

[Bar15] Alexandre Barachant. PyRiemann: Python package for covariance matrices manipulation and Biosignal classification with application in Brain Computer interface, 2015. URL: https://github.com/alexandrebarachant/pyRiemann, doi:10.5281/zenodo.3715511.

[BBL$^+$17] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. URL: https://ieeexplore.ieee.org/document/7974879, doi:10.1109/MSP.2017.2693418.

[BG18] Gary Bécigneul and Octavian-Eugen Ganea. Riemannian Adaptive Optimization Methods. In *Proc. of ICLR 2019*, pages 1–16, 2018. URL: http://arxiv.org/abs/1810.00760.

[BPK10] Marián Boguná, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. *Nature communications*, 1(1):1–8, Oct 2010. URL: https://www.nature.com/articles/ncomms1063, doi:10.1038/ncomms1063.

[CBA15] Emmanuel Chevallier, Frédéric Barbaresco, and Jesus Angulo. Probability density estimation on the hyperbolic space applied to radar processing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9389:753–761, 2015. URL: https://link.springer.com/chapter/10.1007/978-3-319-25040-3_80, doi:10.1007/978-3-319-25040-3_80.

[CCD17] Benjamin Paul Chamberlain, James Clough, and Marc Peter Deisenroth. Neural embeddings of graphs in hyperbolic space. *13th International Workshop on Mining and Learning with Graphs*, 2017. URL: https://arxiv.org/abs/1705.10359.

[Cen12] Andrea Censi. PyGeometry: Library for handling various differentiable manifolds., 2012. URL: https://github.com/AndreaCensi/geometry.

[CS16] Anoop Cherian and Suvrit Sra. Positive Definite Matrices: Data Representation and Applications to Computer Vision. In *Algorithmic Advances in Riemannian Geometry and Applications*. Springer, 2016. URL: https://www.springerprofessional.de/en/positive-definite-matrices-symmetric-positive-definite-spd-matri/10816206, doi:10.1007/978-3-319-45026-1.

[CSS05] Sueli IR Costa, Sandra A Santos, and João E Strapasson. Fisher information matrix and hyperbolic geometry. In *IEEE Information Theory Workshop, 2005.*, pages 3–pp. IEEE, 2005. URL: https://ieeexplore.ieee.org/document/1531851, doi:10.1109/ITW.2005.1531851.

[DKZ09] Ian L. Dryden, Alexey Koloydenko, and Diwei Zhou. Non-Euclidean statistics for covariance matrices, with applications to diffusion tensor imaging. *Annals of Applied Statistics*, 3(3):1102–1123, September 2009. Publisher: Institute of Mathematical Statistics. URL: https://projecteuclid.org/euclid.aoas/1254773280, doi:10.1214/09-AOAS249.

[FHP15] Masoud Faraki, Mehrtash T Harandi, and Fatih Porikli. Material Classification on Symmetric Positive Definite Manifolds. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 749–756, 1 2015. URL: https://ieeexplore.ieee.org/document/7045959, doi:10.1109/WACV.2015.105.

[GBH18] Octavian Ganea, Gary Becigneul, and Thomas Hofmann. Hyperbolic neural networks. In *Advances in Neural Information Processing Systems 31 (NIPS)*, pages 5345–5355. Curran Associates, Inc., 2018. URL: http://papers.nips.cc/paper/7780-hyperbolic-neural-networks.pdf.

[Gro87] Mikhail Gromov. *Hyperbolic Groups*, pages 75–263. Springer New York, New York, NY, 1987. URL: https://link.springer.com/chapter/10.1007/978-1-4613-9586-7_3, doi:10.1007/978-1-4613-9586-7_3.

[GWB$^+$19] Zhi Gao, Yuwei Wu, Xingyuan Bu, Tan Yu, Junsong Yuan, and Yunde Jia. Learning a robust representation via a deep network on symmetric positive definite manifolds. *Pattern Recognition*, 92:1–12, August 2019. URL: https://linkinghub.elsevier.com/retrieve/pii/S0031320319301062, doi:10.1016/j.patcog.2019.03.007.

[GZH$^+$19] Thomas Gerald, Hadi Zaatiti, Hatem Hajri, Nicolas Baskiotis, and Olivier Schwander. From node embedding to community embedding : A hyperbolic approach, 2019. URL: https://arxiv.org/abs/1907.01662, arXiv:1907.01662.

[HHLS16] M. T. Harandi, R. Hartley, B. Lovell, and C. Sanderson. Sparse coding on symmetric positive definite manifolds using bregman divergences. *IEEE Transactions on Neural Networks and Learning Systems*, 27(6):1294–1306, 2016. doi:10.1109/TNNLS.2014.2387383.

[HKKM10] Stephan Huckemann, Peter Kim, Ja Yong Koo, and Axel Munk. Möbius deconvolution on the hyperbolic plane with application to impedance density estimation. *Annals of Statistics*, 38(4):2465–2498, 2010. URL: https://projecteuclid.org/euclid.aos/1278861254, doi:10.1214/09-AOS783.

[HVS$^+$16] Junpyo Hong, Jared Vicory, Jörn Schulz, Martin Styner, J S Marron, and StephenM Pizer. Non-Euclidean Classification of Medically Imaged Objects via s-reps. *Med Image Anal*, 31:37–45, 2016. URL: https://www.sciencedirect.com/science/article/abs/pii/S1361841516000141, doi:10.1016/j.media.2016.01.007.

[JDM12] Sungkyu Jung, Ian L. Dryden, and J. S. Marron. Analysis of principal nested spheres. *Biometrika*, 99(3):551–568, 2012. URL: http://www.statistics.pitt.edu/sungkyu/papers/Biometrika-2012-Jung-551-68.pdf, doi:10.1093/biomet/ass022.

[KH05] John T Kent and Thomas Hamelryck. Using the Fisher-Bingham distribution in stochastic models for protein structure. *Quantitative Biology, Shape Analysis, and Wavelets*, 24(1):57–60, 2005. URL: http://www.amsta.leeds.ac.uk/statistics/workshop/lasr2005/Proceedings/kent.pdf.

[Koc19] Maxim Kochurov. Geoopt: Riemannian Adaptive Optimization Methods with pytorch optim, 2019. URL: https://arxiv.org/abs/2005.02819.

[KPK$^+$10] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82:036106, Sep 2010. URL: https://journals.aps.org/pre/abstract/10.1103/PhysRevE.82.036106, doi:10.1103/PhysRevE.82.036106.

[KS17] Line Kühnel and Stefan Sommer. Computational Anatomy in Theano. *CoRR*, 2017. URL: https://link.springer.com/chapter/10.1007/978-3-319-67675-3_15, doi:10.1007/978-3-319-67675-3_15.

[MGLB$^+$] Nina Miolane, Nicolas Guigui, Alice Le Brigant, Johan Mathe, Benjamin Hou, Yann Thanwerdas, Stefan Heyder, Olivier Peltre, Nicolas Koep, Hadi Zaatiti, Hatem Hajri, Yann Cabanes, Thomas Gerald, Paul Chauchat, Daniel Brooks, Christian Shewmake, Bernhard Kainz, Claire Donnat, Susan Holmes, and Xavier Pennec. Geomstats : a Python Package for Riemannian Geometry in Machine Learning. URL: https://arxiv.org/abs/2004.04667.

[MJK$^+$18] Mayank Meghwanshi, Pratik Jawanpuria, Anoop Kunchukuttan, Hiroyuki Kasai, and Bamdev Mishra. McTorch, a manifold optimization library for deep learning, 2018. URL: http://arxiv.org/abs/1810.01811.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26 (NIPS)*, pages 3111–3119. Curran Associates, Inc., 2013. URL: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality, doi:https://dl.acm.org/doi/10.5555/2999792.2999959.

[NDV+14] Bernard Ng, Martin Dressler, Gaël Varoquaux, Jean Baptiste Poline, Michael Greicius, and Bertrand Thirion. Transport on Riemannian manifold for functional connectivity-based classification. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8674 LNCS, pages 405–412, Cham, 2014. Springer International Publishing. URL: http://link.springer.com/10.1007/978-3-319-10470-6{_}51, doi:10.1007/978-3-319-10470-6_51.

[NK17] Maximillian Nickel and Douwe Kiela. Poincaré Embeddings for Learning Hierarchical Representations. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6338–6347. Curran Associates, Inc., 2017. URL: http://papers.nips.cc/paper/7213-poincare-embeddings-for-learning-hierarchical-representations.pdf.

[PFA06] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A Riemannian Framework for Tensor Computing. *International Journal of Computer Vision*, 66(1):41–66, 1 2006. URL: https://link.springer.com/article/10.1007/s11263-005-3222-z, doi:10.1007/s11263-005-3222-z.

[PKS+12] Fragkiskos Papadopoulos, Maksim Kitsak, M Ángeles Serrano, Marián Boguná, and Dmitri Krioukov. Popularity versus similarity in growing networks. *Nature*, 489(7417):537–540, 2012. URL: https://www.nature.com/articles/nature11459, doi:10.1038/nature11459.

[Pos01] Mikhail Postnikov. *Riemannian Geometry*. Encyclopaedia of Mathem. Sciences. Springer, 2001. URL: https://encyclopediaofmath.org/wiki/Riemannian_geometry, doi:10.1007/978-3-662-04433-9.

[PSF19] Xavier Pennec, Stefan Sommer, and Tom Fletcher. *Riemannian Geometric Statistics in Medical Image Analysis*. Elsevier Ltd, first edit edition, 2019. URL: https://www.elsevier.com/books/riemannian-geometric-statistics-in-medical-image-analysis/pennec/978-0-12-814725-2, doi:10.1016/C2017-0-01561-6.

[SDSGR18] Frederic Sala, Chris De Sa, Albert Gu, and Christopher Re. Representation tradeoffs for hyperbolic embeddings. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4460–4469, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL: http://proceedings.mlr.press/v80/sala18a.html.

[SMA10] Yusuke Shinohara, Takashi Masuko, and Masami Akamine. Covariance clustering on Riemannian manifolds for acoustic model compression. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4326–4329, 3 2010. URL: https://ieeexplore.ieee.org/document/5495661, doi:10.1109/ICASSP.2010.5495661.

[STK05] Olaf Sporns, Giulio Tononi, and Rolf Kötter. The human connectome: A structural description of the human brain. *PLOS Computational Biology*, 1(4):0245–0251, 09 2005. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1239902/, doi:10.1371/journal.pcbi.0010042.

[TKW16] James Townsend, Niklas Koep, and Sebastian Weichwald. Pymanopt: A python toolbox for optimization on manifolds using automatic differentiation. *Journal of Machine Learning Research*, 17(137):1–5, 2016. URL: http://jmlr.org/papers/v17/16-177.html, doi:https://dl.acm.org/doi/10.5555/2946645.3007090.

[WAZF18] Eleanor Wong, Jeffrey S. Anderson, Brandon A. Zielinski, and P. Thomas Fletcher. Riemannian Regression and Classification Models of Brain Networks Applied to Autism. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11083 LNCS of *Lecture Notes in Computer Science*, pages 78–87. Springer International Publishing, 2018. URL: https://link.springer.com/chapter/10.1007/978-3-030-00755-3_9, doi:10.1007/978-3-030-00755-3_9.

[Wyn14] Kieran Wynn. PyQuaternions: A fully featured, pythonic library for representing and using quaternions, 2014. URL: https://github.com/KieranWynn/pyquaternion.

[WZD+13] Jinhui Wang, Xinian Zuo, Zhengjia Dai, Mingrui Xia, Zhilian Zhao, Xiaoling Zhao, Jianping Jia, Ying Han, and Yong He. Disrupted functional brain connectome in individuals at risk for Alzheimer's disease. *Biological Psychiatry*, 73(5):472–481, 2013. URL: http://dx.doi.org/10.1016/j.biopsych.2012.03.026, doi:10.1016/j.biopsych.2012.03.026.

[YZLM12] Ying Yuan, Hongtu Zhu, Weili Lin, and J S Marron. Local polynomial regression for symmetric positive definite matrices. *Journal of the Royal Statistical Society Series B*, 74(4):697–719, 2012. URL: https://econpapers.repec.org/RePEc:bla:jorssb:v:74:y:2012:i:4:p:697-719, doi:10.1111/j.1467-9868.2011.01022.x.