hub-toolbox Documentation

Release 2.3

Roman Feldbauer

Jan 29, 2019

Contents

The Hub Toolbox is a software suite for hubness analysis and hubness reduction in high-dimensional data.

CHAPTER 1

User Guide

The user guide explains how to install the Hub Toolbox, how to analyze your data sets for hubness, and how to use the Hub Toolbox to lift this *curse of dimensionality*.

1.1 Which Hub Toolbox to choose

The Hub Toolbox is available as Python and Matlab scripts. If in doubt, use the Hub Toolbox for Python. See below for a more detailed description.

1.1.1 hub-toolbox-matlab

The Hub Toolbox was originally developed for Matlab/Octave. We still provide these scripts, however, development is limited to bugfixing. No new functionality will be added. The Hub Toolbox for Matlab supports:

- hubness analysis
- hubness reduction
  - Mutual Proximity
  - Local Scaling
  - Shared Nearest Neighbors
- evaluation
  - k-NN classification
  - Goodman-Kruskal index

for distance matrices.

1.1.2 hub-toolbox-python3

The Hub Toolbox for Python3 was initially ported from the Matlab code. Development now focuses on these scripts. It is thus continuously being extended for new functionality and is tested and documented thoroughly. The Hub Toolbox for Python3 offers all the functionality the Matlab scripts offer, plus:

- additional hubness reduction methods
  - centering
  - DisSim
- using similarity matrices instead of distance matrices
- support for sparse matrices (some modules)
- support for parallel processing (some modules)
- performance improvements (some modules)
- unit tests
- this documentation

We recommend using hub-toolbox-python3 for all users. This documentation will assume you are using these scripts.

1.2 Installation

Most of the instructions below assume you are running a Linux system. It might be possible to install the Hub Toolbox on Mac or Windows systems. We cannot, however, give any guidance for these cases at this point.

1.2.1 Prerequisites

Python

The Hub Toolbox currently requires Python 3.6 or higher. You can check this on your system with:

```
python3 --version
```

If Python3 is missing, or its version is lower than 3.6, please install it via the package manager of your operating system (e.g. `apt` in Debian/Ubuntu or `dnf` in Fedora).

You might also consider using the Anaconda environment for easy Python environment and package handling.

numpy/scipy/scikit-learn

The Hub Toolbox heavily relies on numpy and requires scipy and scikit-learn for some functions. Please install these packages via your operating system's package manager (e.g. `sudo apt install python3-numpy python3-scipy python3-sklearn`) or use Anaconda: `conda install numpy scipy scikit-learn`. We do not recommend installation via `pip` since this may lead to suboptimal performance unless configured properly.

1.2.2 Stable Hub Toolbox release

Stable releases of the Hub Toolbox are added to PyPI . To install the latest stable release, simply use *pip* (you may need to install it first via your operating system's package manager, e.g. `sudo apt install python3-pip`).

```
pip3 install hub-toolbox
```

Alternatively, you may download the latest release from GitHub and follow the instructions of a development installation (from source) below, omitting the `git clone` step.

1.2.3 Installation from source

For a bleeding edge version of the Hub Toolbox, you can install it from the latest sources: On the console, change to the directory, under which the Hub Toolbox should be installed. Then obtain a copy of the latest sources from GitHub:

```
git clone https://github.com/OFAI/hub-toolbox-python3.git
```

They will be cloned to a subdirectory called `hub-toolbox-python3`. The Hub Toolbox must then be built and installed with

```
cd hub-toolbox-python3
python3 setup.py build
sudo python3 setup.py install
```

The Hub Toolbox is now available system wide. Optionally, you can now run a test suite by

```
sudo python3 setup.py test
```

If this prints an `OK` message, you are ready to go. Note, that some skipped tests are fine.

1.3 Tutorial

In this tutorial you will analyze the dexter dataset for hubness, reduce hubness, and observe how this improves internal and external evaluation measures.

From there on you will be able to apply the techniques offered by the Hub Toolbox to the dataset of your choice.

1.3.1 Prerequisites

For this tutorial, you will require a working installation of the Hub Toolbox. If you don't have one yet, please follow the instructions in *Installation*.

1.3.2 Analyze the dexter dataset

The Hub Toolbox ships with DEXTER as an example dataset. DEXTER is a text classification problem in a bag-of-word representation. This is a binary classification problem with sparse continuous input variables. This dataset was one of five datasets of the NIPS 2003 feature selection challenge. For more info, see: http://archive.ics.uci.edu/ml/datasets/Dexter

On the terminal, start a Python shell:

```
python3
```

Consider using an IPython/jupyter notebook as a more flexible and powerful alternative.

The `HubnessAnalysis` class automatically analyzes the DEXTER example dataset, if invoked without further parameters:

```
>>> from hub_toolbox.HubnessAnalysis import HubnessAnalysis
>>> ana = HubnessAnalysis()
>>> ana.analyze_hubness()
```

This will print a rather lengthy result log:

```
NO PARAMETERS GIVEN! Loading & evaluating DEXTER data set.
DEXTER is a text classification problem in a bag-of-word
representation. This is a two-class classification problem
with sparse continuous input variables.
This dataset is one of five datasets of the NIPS 2003
feature selection challenge.
http://archive.ics.uci.edu/ml/datasets/Dexter


================
Hubness Analysis
================

ORIGINAL DATA:
data set hubness (S^k= 5)                : 4.22
% of anti-hubs at k= 5                   : 26.67%
% of k= 5-NN lists the largest hub occurs: 23.67%
data set hubness (S^k=10)                : 3.98
% of anti-hubs at k=10                   : 17.67%
% of k=10-NN lists the largest hub occurs: 50.0%
k= 1-NN classification accuracy          : 80.33%
k= 5-NN classification accuracy          : 80.33%
k=20-NN classification accuracy          : 84.33%
Goodman-Kruskal index (higher=better)    : 0.104
embedding dimensionality                 : 20000
intrinsic dimensionality estimate        : 161

MUTUAL PROXIMITY (Empiric):
data set hubness (S^k= 5)                : 0.712
% of anti-hubs at k= 5                   : 3.0%
% of k= 5-NN lists the largest hub occurs: 6.0%
data set hubness (S^k=10)                : 0.71
% of anti-hubs at k=10                   : 0.0%
% of k=10-NN lists the largest hub occurs: 10.67%
k= 1-NN classification accuracy          : 82.67%
k= 5-NN classification accuracy          : 89.67%
k=20-NN classification accuracy          : 88.67%
Goodman-Kruskal index (higher=better)    : 0.132
embedding dimensionality                 : 20000
intrinsic dimensionality estimate        : 161

MUTUAL PROXIMITY (Independent Gaussians):
data set hubness (S^k= 5)                : 0.805
% of anti-hubs at k= 5                   : 4.667%
% of k= 5-NN lists the largest hub occurs: 5.667%
```

(continues on next page)

(continued from previous page)

```
data set hubness (S^k=10)                : 1.21
% of anti-hubs at k=10                   : 0.0%
% of k=10-NN lists the largest hub occurs: 12.67%
k= 1-NN classification accuracy          : 83.67%
k= 5-NN classification accuracy          : 89.0%
k=20-NN classification accuracy          : 90.0%
Goodman-Kruskal index (higher=better)    : 0.135
embedding dimensionality                 : 20000
intrinsic dimensionality estimate        : 161

LOCAL SCALING (NICDM):
parameter k = 7 (for optimization use the individual modules of the HUB-TOOLBOX)
data set hubness (S^k= 5)                : 2.1
% of anti-hubs at k= 5                   : 0.6667%
% of k= 5-NN lists the largest hub occurs: 8.667%
data set hubness (S^k=10)                : 1.74
% of anti-hubs at k=10                   : 0.0%
% of k=10-NN lists the largest hub occurs: 16.0%
k= 1-NN classification accuracy          : 84.67%
k= 5-NN classification accuracy          : 85.0%
k=20-NN classification accuracy          : 85.0%
Goodman-Kruskal index (higher=better)    : 0.118
embedding dimensionality                 : 20000
intrinsic dimensionality estimate        : 161

CENTERING:
data set hubness (S^k= 5)                : 1.62
% of anti-hubs at k= 5                   : 6.667%
% of k= 5-NN lists the largest hub occurs: 8.333%
data set hubness (S^k=10)                : 1.38
% of anti-hubs at k=10                   : 1.333%
% of k=10-NN lists the largest hub occurs: 13.0%
k= 1-NN classification accuracy          : 85.0%
k= 5-NN classification accuracy          : 87.67%
k=20-NN classification accuracy          : 89.33%
Goodman-Kruskal index (higher=better)    : 0.19
embedding dimensionality                 : 20000
intrinsic dimensionality estimate        : 161

DISSIM GLOBAL:
data set hubness (S^k= 5)                : 1.87
% of anti-hubs at k= 5                   : 6.333%
% of k= 5-NN lists the largest hub occurs: 8.667%
data set hubness (S^k=10)                : 1.62
% of anti-hubs at k=10                   : 1.667%
% of k=10-NN lists the largest hub occurs: 14.67%
k= 1-NN classification accuracy          : 84.0%
k= 5-NN classification accuracy          : 88.67%
k=20-NN classification accuracy          : 88.67%
Goodman-Kruskal index (higher=better)    : 0.189
embedding dimensionality                 : 20000
intrinsic dimensionality estimate        : 161
```

1.3.3 Interpreting the results

Let us dissect these results: The first block appears, because we did not provide any parameters, when instantiating `HubnessAnalysis`. It thus goes into example mode and tells you a little bit about the dataset being used.

The actual results of the analysis are grouped into blocks by experiments. Here, an experiment comprises the following:

1. a hubness reduction method is applied to the dataset's distance matrix to obtain a matrix of secondary distances (except for centering, which changes vector data)
2. hubness and additional measures of hubs and anti-hubs are calculated (in this case twice, for two different neighborhood sizes)
3. k-nearest neighbor classification leave-one-out cross-validation is performed (in this case three times, for three different values of k)
4. the Goodman-Kruskal index is calculated for the secondary distance matrix

Additionally, the intrinsic dimension is estimated once for the dataset for all experiments.

The second block (under the *Hubness Analysis* headline) is the experiment using primary distances. For text-based datasets like DEXTER cosine distances are used frequently. We observe considerable hubness of `S^(k=5) = 4.22`. (As a rule of thumb, consider values above `1.2` as 'high hubness'). Knowing that hubness is a phenomenon of intrinsically high dimensional data, it is not surprising that the intrinsic dimension estimate of `161` is also considerably high (although much lower than the embedding dimension of `20000`). We also observe a lot of anti-hubs (i.e. points that are not among the k-nearest neighbors of any other point; or in other words: their `k-occurence=0`), while the largest hub is among the k-nearest neighbors of very many points. We find k-NN classification accuracy of roughly `80%`.

The third block contains the results of an Mutual Proximity experiment, using the empirical distance distribution to rescale these distances. We observe tremendously reduced hubness, hardly any anti-hubs, and reduced k-occurence of the largest hub. Also, internal evaluation with the Goodman-Kruskal index improves compared to using the primary distances. Mutual Proximity is thus able to reduce hubness, but we don't know yet, whether these secondary distances still reflect the semantics of the dataset. Looking at the k-NN classification, it seems like these were actually improved, because accuracy increased to nearly `90%`. Note that embedding and intrinsic dimension do not change, because they are computed on the original dataset.

The following blocks represent other hubness reduction methods, some performing as well as Mutual Proximity, some performing worse. However, all of them improve internal as well as external evaluation measures.

1.3.4 Analyzing other datasets

`HubnessAnalysis` can also be used to investigate other datasets. You will require at least a numpy array of your feature vectors (called *vectors*), or a distance matrix `D` (where `D[i, j]` is the distance between your `i-th` and `j-th` feature vector). If you want to perform classification, you also need to provide a vector with integer labels for each data point (`target` or 'ground-truth'). If you don't have a distance matrix yet, you can use the methods from `Distances` to create one based on euclidean or cosine distances. For other types of distances, you can also use scipy.spatial.distance.pdist.

Now simply call

```
>>> from hub_toolbox.HubnessAnalysis import HubnessAnalysis
>>> ana = HubnessAnalysis(D, vectors, target)
>>> ana.analyze_hubness(experiments="orig,mp,nicdm,dsg",
                        hubness_k=(5, 10), knn_k=(10, 20))
```

Note, how we provided parameters to `analyze_hubness`: The Hub Toolbox will now perform four experiments (original data, Mutual Proximity (Empiric), Local Scaling (NICDM), and DisSim Global). The neighborhood size is

the same as in the last example, but we changed the classification to 10-NN and 20-NN (instead of 1-NN, 5-NN, and 20-NN).

Looking at your output, you may notice a line that was not discussed before: *NICDM* has a parameter *k* that can be tuned. Other methods do so as well. The convenience class `HubnessAnalysis` does not allow to change the default values for the methods' parameters. To do so, you can use the individual methods of the Hub Toolbox directly, which will be covered in the next section.

1.3.5 Using individual methods

In this section we will revisit the analysis we performed previously on the DEXTER dataset. This time, instead of using the convenience class `HubnessAnalysis`, we will employ the individual modules of the Hub Toolbox in order to see, how to use it in a more flexible way.

Loading the example dataset

```
>>> from hub_toolbox.IO import load_dexter
>>> D, labels, vectors = load_dexter()
>>> vectors.shape
(300, 20000)
```

We see that DEXTER comprises `300` points in an embedding dimension of `20000`. The *IntrinsicDim* module can provide some insight, how well this reflects the 'true' dimensionality of the dataset, by

Calculating an intrinsic dimension estimate

```
>>> from hub_toolbox.IntrinsicDim import intrinsic_dimension
>>> intrinsic_dimension(vectors, k1=6, k2=12, estimator='levina', trafo=None)
74
```

The MLE by Levina and Bickel with neighborhood `[6, 12]` tells us that the intrinsic dimension is much lower than the embedding dimension, but is still considerably high. We can assume, that this dataset is prone to

Hubness

```
>>> from hub_toolbox.Hubness import hubness
>>> S_k, D_k, N_k = hubness(D=D, k=5, metric='distance')
>>> print("Hubness:", S_k)
Hubness: 4.222131665788378
```

Besides the hubness in `S_k`, you also get the objects `D_k` and `N_k`, which contain the `k` nearest neighbors of all elements and the n-occurence, respectively. From them you can extract more detailed information about hubs and anti-hubs.

External and internal evaluation can be performed with the following methods:

k-NN classification

```
>>> from hub_toolbox.KnnClassification import score
>>> acc, corr, cmat = score(D=D, target=labels, k=[1,5], metric='distance')
>>> print("k=5-NN accuracy:", acc[1, 0])
k=5-NN accuracy: 0.803333333333
```

Also in this case, you obtain three objects: `acc` contains the accuracy values, `corr` contains information about each point, whether it was classified correctly or not, and `cmat` contains the corresponding confusion matrices. All three objects contain their information of each k-NN experiment defined with parameter `k=[1,5]`.

Goodman-Kruskal index

```
>>> from hub_toolbox.GoodmanKruskal import goodman_kruskal_index
>>> gamma = goodman_kruskal_index(D=D, classes=labels, metric='distance')
>>> print("Goodman-Kruskal index:", gamma)
Goodman-Kruskal index: 0.103701886155
```

Calculating the `Goodman-Kruskal index` is straight forward.

Hubness reduction

```
>>> from hub_toolbox.MutualProximity import mutual_proximity_empiric
>>> D_mp = mutual_proximity_empiric(D=D, metric='distance')
```

```
>>> from hub_toolbox.LocalScaling import nicdm
>>> D_nicdm = nicdm(D=D, k=10, metric='distance')
```

You now have two objects `D_mp` and `D_nicdm` which contain secondary distances of the DEXTER dataset, rescaled with Mutual Proximity (Empiric) and Local Scaling (NICDM), respectively. They can now be used just as illustrated above for k-NN classification, hubness calculation etc.

The Hub Toolbox provides more methods for hubness reduction than these two, and additional ones will be integrated as they are developed by the hubness community. To see, which methods are currently included, try

```
>>> from hub_toolbox.HubnessAnalysis import SEC_DIST
>>> for k, v in SEC_DIST.items():
...   print(k)
...
dsl
snn
wcent
lcent
mp_gaussi
mp
orig
mp_gauss
nicdm
dsg
cent
ls
mp_gammai
```

The values `v` in this dictionary are actually the hubness reduction functions, so you may invoke them for example like this:

```
>>> D_snn = SEC_DIST['snn'](D)
```

to obtain shared nearest neighbor distances.

Approximate hubness reduction

TODO

For now, please consider the docstrings. If in doubt, please don't hesitate to contact the author.

CHAPTER 2

API Reference

Find all the information about specific modules and functions of the Hub Toolbox in this section.

- genindex
- modindex