ECODOPS
Efficient COverage of Data On Parameter Space

User Manual – Rel. 0.4

1 Introduction

In addition to optimizing the Parameter Space (PS), by selecting all relevant features, we need to make sure that in any classification experiment the training data cover the feature space sufficiently well for the classification in the inference data to result qualitatively acceptable.

Machine Learning (ML) models can be interpreted as complex decision boundaries in the training feature space. The models are expected to learn true patterns, which should then extend their applicability to new datasets. However, for the points that lie outside of the original region of feature space for which decision boundaries were created, model predictions may implement a classification function extrapolated from the training data, which may then not agree with the patterns outside of the training set. The most straightforward solution is to simply match the inference dataset to the training sample. Simple and commonly used solutions could consist into cutting data at some magnitude-limit, or, by working in the colour space, at some colour level.

Cuts performed in single features allow for a better match between the training and inference set in these particular dimensions. However, as the final classification is performed in a space of significantly larger dimensionality, the usefulness of such an approach can be rather limited. A match among individual features does not have to imply proper coverage of the full feature space. A simple counterexample is a 2D square covered by data points drawn from a 2D Gaussian distribution and separated into two subsets by a diagonal. In such case, the histograms of single features show overlap of data in individual dimensions, while in fact there is no data from two subsets overlapping in 2D at all. Therefore, we look in more detail at coverage in the multidimensional feature space of the training and inference data. This is done by projecting the feature space onto two dimensions using the t-SNE[1] method.

There are many ways of mapping N-dimensional feature spaces onto 2D projections. A popular one in astronomy is Self Organizing Map (SOM, Kohonen 1997), and a relevant example of its usage is mapping of multi-color space to visualize which regions are not covered by spectroscopic redshifts (Masters et al. 2015). Here, we use another advanced visualization method, the t-distributed stochastic neighbor embedding (t-SNE, van der Maaten & Hinton 2008), which finds complex nonlinear structures and creates a projection onto an abstract low-dimensional space. Its biggest advantage over other methods is that t-SNE can be used on a feature space of even several thousand dimensions and still create a meaningful 2D embedding. Moreover, unlike in SOM where data points are mapped to cells gathering many observations each, in t-SNE every point from the N-dimensional feature space is represented as a single point of the low-dimensional projection. This makes t-SNE much more precise, allowing to plot the exact data point values over visualized points as different colors or shapes, making the algorithm output easier to interpret. Some disadvantages of using t-

[1] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

SNE are its relatively long computing time and its inability to map new sources added to a dataset after the transformation process, without running the algorithm again.

The t-SNE method makes use of the Kullback-Leibler (K-L) divergence cost function to find an optimal 2D representation of a multi-dimensional space. Through a cyclic process, the method tries to minimize the K-L divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.
Main question is: **How to project data to a subspace by preserving discriminability?**
From the Theory of Information, it is known that not all random events are equally random (see plot below as examples).

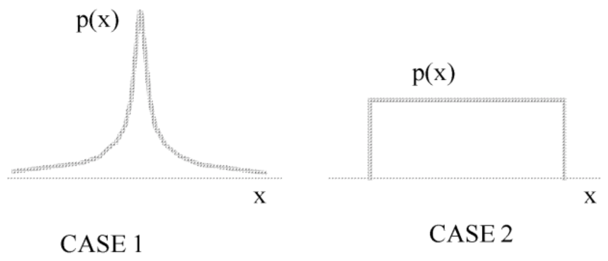


Figure 1 – Examples of random event distributions not equally random.

To quantify this fact, Shannon proposed the concept of Information Entropy:

$$S_H = -log_2 p_X(x) \quad \rightarrow \quad E[S_H] = H(P) = \sum p_X(x) S_H$$

From this concept, two properties were derived, the mutual information $I(x,y)$ and the K-L Divergence $K(f,g)$:

$$I(x,y) = H(x) + H(y) - H(x,y) = H(x) - H(x|y) = H(y) - H(y|x)$$
$$K(f,g) = \int f(x) log\left(\frac{f(x)}{g(x)}\right) dx$$

from which it derives that I(x,y) can be formulated in terms of K-L Divergence (see also Figure 2):

$$I(x,y) = \iint f_{xy}(x,y) log\left(f_{xy}(x,y)/f_x(x)f_y(y)\right) dxdy$$

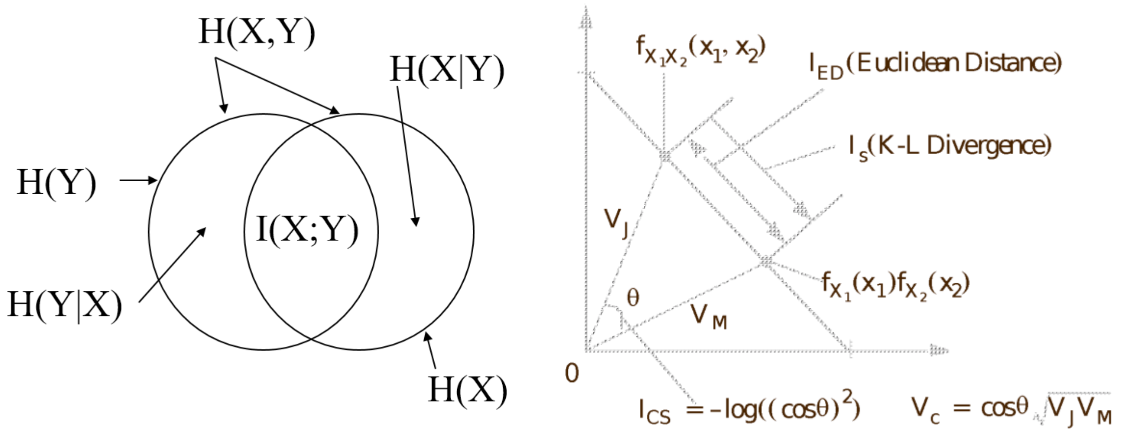

H(X,Y)
H(X|Y)
H(Y)
I(X;Y)
H(Y|X)
H(X)
$f_{X_1X_2}(x_1, x_2)$
I_{ED}(Euclidean Distance)
I_s(K-L Divergence)
V_J
$f_{X_1}(x_1)f_{X_2}(x_2)$
θ
V_M
0
$I_{CS} = -\log((\cos\theta)^2)$
$V_c = \cos\theta\sqrt{V_J V_M}$

Figure 2 – Mutual Information in terms of theory of sets (left) and its geometrical explanation (right).

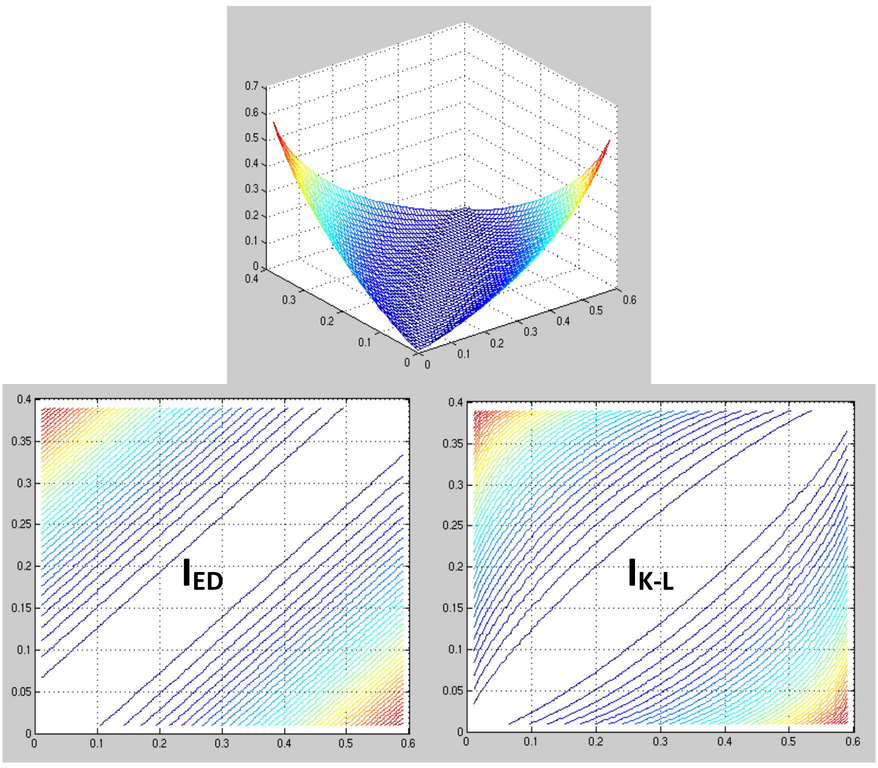
In a statistical sense, the entropy became one of the central "moments" of a distribution, after mean and variance. In such context, the K-L divergence measures the "distance" between PDFs of two distributions f() and g():

$$D_{K-L}(f,g) = \int f(x) log\left(\frac{f(x)}{g(x)}\right) dx \qquad (1)$$

With K-L Divergence, we can thus calculate how much information is lost, when we approximate one distribution with another, in a more realistic way than through Euclidean metric.
Essentially, what we are looking at, with the K-L Divergence, is the expectation of the log difference between the probability of data in the original distribution with the approximating distribution. Again, if we think in terms of binary log, we can interpret this as "how many bits of information we expect to lose". In fact, through the properties of logarithms, we can rewrite the above formula (1) in terms of expectation:

$$D_{K-L}(p,q) = E[logp(x) - logq(x)]$$

There is indeed an important difference between the Information entropy as measured in euclidean terms and as K-L divergence, easily derivable from the following plot example (Figure 3).



(Euclidean) (K-L Divergence)

Figure 3 – Example of difference between the Mutual Information calculated for a generic function (top panel) from Euclidean distance (bottom left panel) or from K-L) Divergence distance (bottom right panel).

Now we are able to give the answer to the initial question: **the answer is by maximizing the Mutual Information between desired responses and the output of the (nonlinear) mapper**. This process is done by implicitly solving the feature extraction that maximize the classification performance, or through the classical PCA (Principal Component Analysis) or through the random embedding (selectable via a user-selected parameter; see Section 5 for details about the parameter "init").

2 Installation

The TSNE package is composed by the following files:

1. TSNE.py: the python script;
2. Config.ini: the configuration file containing all the model parameters and I/O setup information.

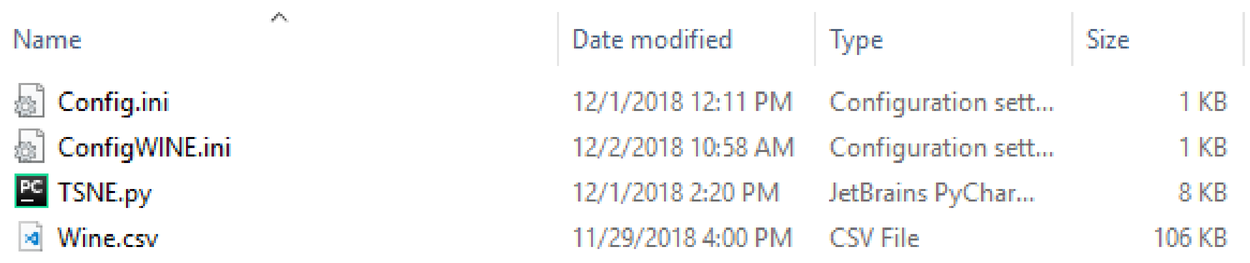
Before to launch the python script, the following python packages must be installed, together with a python version 3.7.1 or higher:

1. numpy
2. ConfigObj
3. argparse
4. sklearn
5. pandas
6. matplotlib
7. csv
8. astropy

Make sure that Python has been added to PATH in order to execute the TSNE.py script from any arbitrary path on your machine.

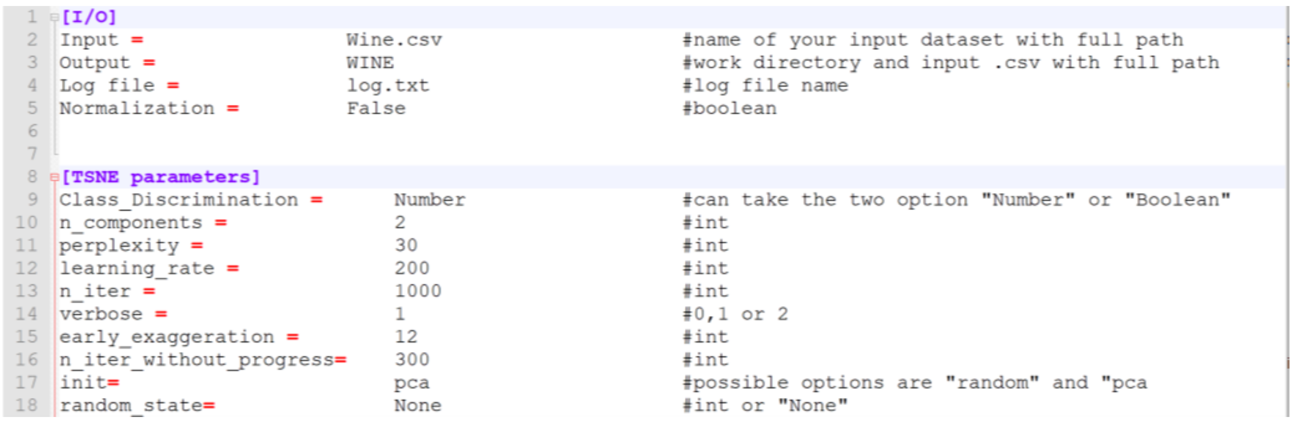
3 How to

In order to use the tool you have to put in the same work directory the TSNE.py script, your own copy of the Config.ini file or the original Config.ini file and your input dataset, properly formatted as a .csv or .fits file (please see Section 4 for details) .
As an example, your work directory and Config.ini file should look like this:



Name	Date modified	Type	Size
Config.ini	12/1/2018 12:11 PM	Configuration sett...	1 KB
ConfigWINE.ini	12/2/2018 10:58 AM	Configuration sett...	1 KB
TSNE.py	12/1/2018 2:20 PM	JetBrains PyChar...	8 KB
Wine.csv	11/29/2018 4:00 PM	CSV File	106 KB

Figure 4 – Typical content of the installation folder.



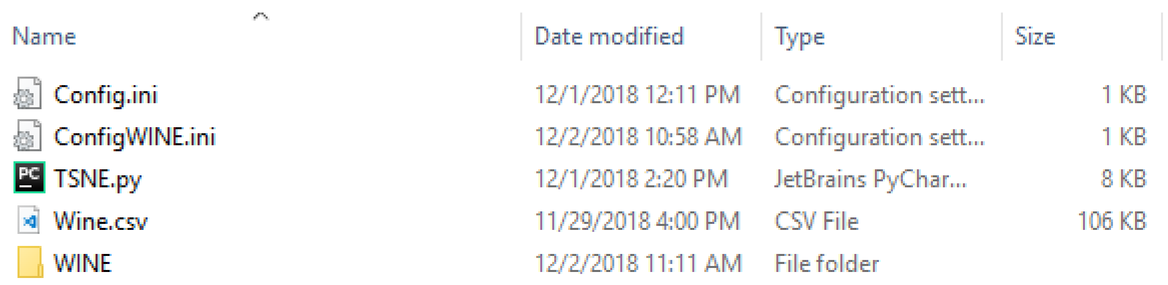
```
[I/O]
Input =                 Wine.csv                    #name of your input dataset with full path
Output =                WINE                        #work directory and input .csv with full path
Log file =              log.txt                     #log file name
Normalization =         False                       #boolean


[TSNE parameters]
Class_Discrimination =      Number                  #can take the two option "Number" or "Boolean"
n_components =              2                       #int
perplexity =                30                      #int
learning_rate =             200                     #int
n_iter =                    1000                    #int
verbose =                   1                       #0,1 or 2
early_exaggeration =        12                      #int
n_iter_without_progress=    300                     #int
init=                       pca                     #possible options are "random" and "pca
random_state=               None                    #int or "None"
```

Figure 5 – Internal structure of the default config.ini setup file.

The script will create a sub-directory (in this case named WINE) in the current directory or create a new one if the directory already exists. In this sub-directory the script will store the log.txt, the Output.csv and the Scatter plot (see Section 4).
After this early preparation, open the Command Prompt (Windows) or the Terminal (Linux) and reach the path where you installed the package files. To execute the TSNE please give the following commands and press enter:

1. **`python TSNE.py`** if you want to use the default Config.ini or
2. **`python TSNE.py -c ConfigWINE.ini`** if you want to use your own copy of the Config.ini (in this case named ConfigWINE.ini).



Name	Date modified	Type	Size
Config.ini	12/1/2018 12:11 PM	Configuration sett...	1 KB
ConfigWINE.ini	12/2/2018 10:58 AM	Configuration sett...	1 KB
TSNE.py	12/1/2018 2:20 PM	JetBrains PyChar...	8 KB
Wine.csv	11/29/2018 4:00 PM	CSV File	106 KB
WINE	12/2/2018 11:11 AM	File folder	

Figure 6 – Working directory after the execution of the WINE project.

In this image you can see the work directory after the TSNE.py execution with the new Output folder named WINE (as stated in the ConfigWINE.ini). In the WINE folder you will find the outputs as described in section 4.

4 I/O

4.1 Input Data Format

The input file must be a .csv (*Comma Separated Values*) with a header or a .fits file. The first column should be the ID column, while the last column must be the class label column. If you have selected "Boolean" as Class_Discrimination parameter in the Config.ini, this column has to be filled with *True* and *False* strings labelling the two classes. If, instead, you have selected "Number", the column has to be filled with a number of unique integers equal to the value of the n_classes parameter (for example: 0 for class 1, 1 for class 2, 2 for class 3, etc.).

4.2 Output

The method will output in the selected folder an *Output.csv* file, a log file *<name>.txt* and the scatter plot of of the data in the bi-dimensional embedded space.
The *Output.csv* file contains three columns, i.e. the **class** of each object as in the original input file and the **x** and **y** columns containing the object's coordinates in the bi-dimensional embedded space.
The *log.txt* file contains a copy of the user configuration file, information on the duration of the execution, the number of classes (**n_classes**) and dimensions (**n_dimensions**) of the input file, the obtained final Kullback - Leibler divergence (**kl_divergence**) and a copy of the t-SNE output statistics.
The output statistic information provided by the method is a string message like this:

```
[t-SNE] Iteration 3000: error = 1.1746874, gradient norm = 0.0000192 (50 iterations in 9.008s)
```

This information includes:

- iteration number;
- error: Kullback-Leibler (K-L) divergence after optimization (error on the minimization of the K-L divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data);
- gradient norm: normalized first derivative of the error;
- time duration of the last 50 iterations.

5 Configuration of an experiment

The input, output and parameters of the method are controlled via the **Config.ini** file. This is divided in two subsections, [I/O] and [model parameters].

[I/O] must contain the following information:

- **Input**: the name of the .csv (or .fits) input dataset. If the .csv (or .fits) is in a different folder with respect to the TSNE.py, please insert the full path of the .csv (or .fits);
- **Output**: the output folder name, if a folder with this name already exists in the specified path, a new folder will be created in the same path, using current date and time as suffix;
- **Log file**: the name of the output log .txt file that will contain the model parameters and the final measured Kullback-Leibler divergence.
- **Normalization:** this parameter can take the values "True" or "False", use "True" if you want to normalize the features before running the t-SNE, "False" otherwise. The normalization performs the following operation on each feature X of the input dataset, in order to normalize it in the interval [0, 1]:
  ```
  X_std = (X - X.min) / (X.max - X.min)
  X_scaled = X_std * (max - min) + min
  ```

  where X.min and X.max are, respectively, the minimum and maximum value assumed by the feature X within the input data.

[TSNE parameters] must contain the following information:

- **Class_Discrimination**: this parameter can take two values, "*Boolean*" or "*Number*". Insert "*Boolean*" if you have a two-class problem and the class identifier assumes standard *True* / *False* labels, otherwise you have to impose such two labels. Insert "*Number*" if you have an arbitrary number of classes. In this last case, the class identifiers must be a collection of integer numbers;

- **n_components**: this is the dimension of the embedded lower dimensional space. Leave this parameter equal to 2, unless you want to modify the method for 3-D embedding;
- **perplexity**: the perplexity is related to the number of nearest neighbors used. Larger or denser datasets usually require a larger perplexity. Consider selecting a value between 5 and 50;
- **learning_rate**: the learning rate for t-SNE is usually in the range [10.0, 1000.0]. If the learning rate is too high, the data may look like a 'ball' with any point approximately equidistant from its nearest neighbours. If the learning rate is too low, most points may look compressed in a dense cloud with few outliers. If the cost function gets stuck in a bad local minimum, increasing the learning rate may help;
- **n_iter**: this is the maximum number of iterations for the optimization. Should be at least 250, we suggest a number between 1000 and 5000;
- **verbose:** verbosity level of the method, it can assume the values 0,1 or 2. ; 0 will display no information during the script execution, 1 will display the t-SNE error after the first 250 iterations, while the value 2 will display the t-SNE error every 50 iterations.
- **early_exaggeration** (default value = 12): this parameter controls how tight natural clusters in the original space are in the embedded space and how much space will be left between them (multiplicative factor). For larger values, the space between natural clusters will be larger in the embedded space. This parameter is not very critical. If the cost function increases during initial optimization, the early exaggeration factor or the learning rate might be too high.
- **n_iter_without_progress**: this is the maximum number of iterations without progress before the method aborts the optimization, used after 250 initial iterations with early exaggeration. Note that progress is only checked every 50 iterations, so this value is rounded to the next multiple of 50;
- **init**: Initialization of embedding. Possible options are "random" and "pca". PCA initialization cannot be used with precomputed distances and is usually more globally stable than random initialization;
- **random_state**: this parameter can take the values "int" or "None". If "int" is selected, random_state is the seed used by the random number generator; If None is selected, the random number generator is the RandomState instance used internally by the python function *np.random*. Note that different initializations might result in different local minima of the cost function.

6 Tooltips

Because each run of the TSNE.py script will bring slightly different results (even with the same parameters in the Config.ini), the first problem we want to address is **how we can assess the quality of the visualization?** Our experience is that the best visualization is always obtained with the lowest Kullback-Leibler divergence reported in the log.txt (kl_divergence). So we encourage the user to run the method multiple time (even variating the parameters) and selecting the run with the lowest reported kl_divergence.

There are four parameters that control the optimization of t-SNE and the quality of the resulting embedding:

- perplexity
- early exaggeration factor
- learning rate
- maximum number of iterations

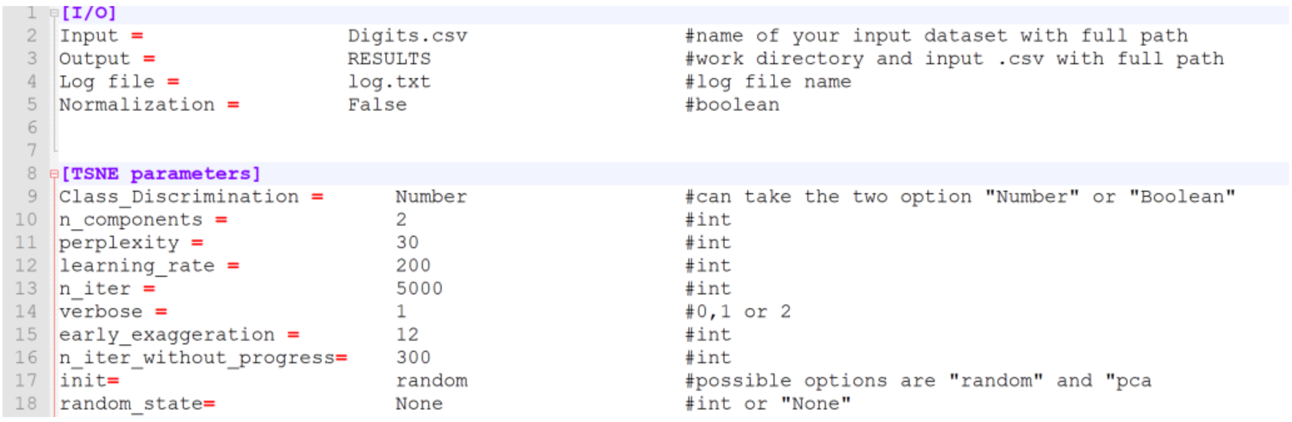
The **perplexity**, as stated in the manual, should take a value between 5 and 50 and it depends strongly on the density of your data. Perplexity is defined as 2 to the power of the Shannon entropy and can be thought as a limit that sets the number of effective nearest neighbors. We advice to increase the perplexity until the points in the embedded space seem uniformly distributed. That is the “*signal”* that the perplexity is too high and you may lower it to find its most appropriate value.

The **maximum number of iterations** is usually high enough and does not need any tuning. The optimization consists of two phases: the early exaggeration phase and the final optimization. During early exaggeration the joint probabilities in the original space will be artificially increased by multiplication with a given factor (**early exaggeration facto**r). Larger factors result in larger gaps between natural clusters in the data. If the factor is too high, the kl_divergence could increase during this phase (and you can check by setting **verbose** = 2 in your Config.ini). A critical parameter is the **learning rate**. If it is too low, gradient descent will get stuck in a bad local minimum. If it is too high the kl_divergence will increase during optimization.

If you have set all these parameters to get the lowest kl_divergence possible and still the visualization in embedded space does not meet you “expectancies”, as a last resort you can set **Normalization** to True in the Config.ini or directly divide your data by a big number and restart the optimization process from the beginning.

7 Example

Here we want to present a full example of the TSNE on the Pen-Based Recognition of Handwritten Digits Data Set (Digits). The dataset is a .csv file, hereafter called Digits.csv, containing 66 columns of which the first one, called “ID”, contains the object’s identification numbers and the last one, named “Class”, contains 6 class unique numerical identifiers. The Digits.csv dataset is placed within the directory containing the TSNE.py script and a copy of the Config.ini file called ConfigDIGIT.ini, whom content is shown in the figure below:



```
[I/O]
Input =                 Digits.csv              #name of your input dataset with full path
Output =                RESULTS                 #work directory and input .csv with full path
Log file =              log.txt                 #log file name
Normalization =         False                   #boolean


[TSNE parameters]
Class_Discrimination =      Number              #can take the two option "Number" or "Boolean"
n_components =              2                   #int
perplexity =                30                  #int
learning_rate =             200                 #int
n_iter =                    5000                #int
verbose =                   1                   #0,1 or 2
early_exaggeration =        12                  #int
n_iter_without_progress=    300                 #int
init=                       random              #possible options are "random" and "pca
random_state=               None                #int or "None"
```

Figure 7 – Configuration of setup file config.ini to execute the experiment.

It is possible to execute the TSNE.py in a Command Prompt (Windows) or Terminal (Unix) with the following command:

```
python TSNE.py -c ConfigDIGITS.ini
```

obtaining the following output files in the RESULTS folder:

1. **log.txt**

```
TSNE log file
TSNE Starting Time: 2018-12-03 16:17:47.620889

*****************   CONFIG FILE COPY   ******************

[I/O]
Input = Digits.csv
Output = E:\TSNE\DIGITS\RESULTS
Log file = log.txt
Normalization = False

[TSNE parameters]
angle = 0.5
early_exaggeration = 12.0
init = random
learning_rate = 200.0
method = barnes_hut
metric = euclidean
min_grad_norm = 1e-07
n_components = 2
n_iter = 5000
n_iter_without_progress = 300
perplexity = 30.0
random_state = True
verbose = 1
kl_divergence = 0.5797737836837769
n_classes = 6
n_dimensions = 64

*****************   END CONFIG FILE COPY   ******************

*****************   DURATION SUMMARY   ******************

TSNE End Time: 2018-12-03 16:18:21.757209
TSNE running time: --- 34.47291827201843 seconds ---

*****************   END DURATION SUMMARY   ******************

*****************   TSNE VERBOSE DUMP   ******************

[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 1083 samples in 0.004s...
[t-SNE] Computed neighbors for 1083 samples in 0.190s...
[t-SNE] Computed conditional probabilities for sample 1000 / 1083
[t-SNE] Computed conditional probabilities for sample 1083 / 1083
[t-SNE] Mean sigma: 8.151373
[t-SNE] KL divergence after 250 iterations with early exaggeration:
56.661419
[t-SNE] KL divergence after 5000 iterations: 0.579774
```

2. **Output.csv** as described in section 4;
3. the scatter plot of the objects in the embedded bi-dimensional parameter space

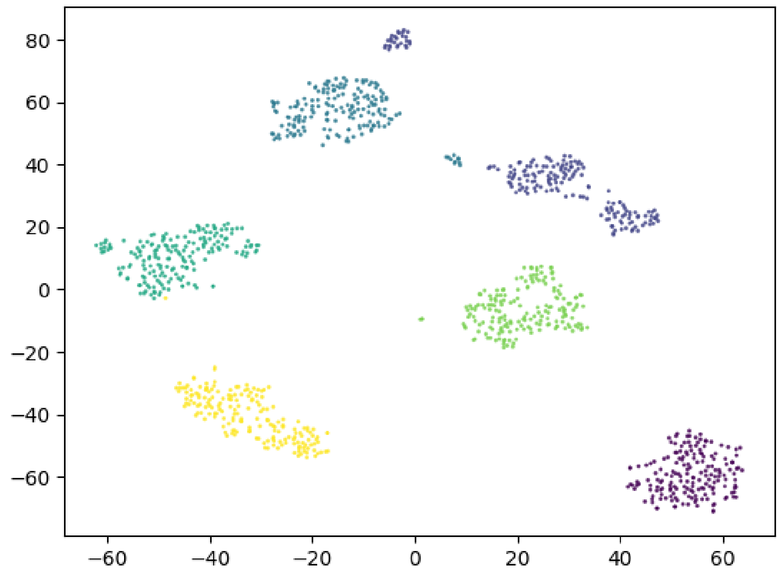


Figure 8 – Output bi-dimensional plot of the method.

As you can see, each of the class has been colored and they appear as separate clusters in the bi-dimensional space.

For any request of more information and help, please contact M. Brescia (massimo.brescia@inaf.it)

8 References

Kohonen, T., ed. 1997, Self-organizing Maps (Berlin, Heidelberg: Springer-Verlag)
Masters, D., Capak, P., Stern, D., et al. 2015, ApJ, 813, 53
van der Maaten, L. & Hinton, G. 2008, Journal of Machine Learning Research, 9, 2579

__oOo__