Deep Triplet-Driven Semi-supervised Embedding Clustering

Deep Triplet-Driven Semi-Supervised Embedding Clustering

Dino Ienco[1] and Ruggero G. Pensa[2]

[1] IRSTEA, UMR TETIS, LIRMM, Univ. Montpellier, Montpellier, France
dino.ienco@irstea.fr
[2] University of Turin, Dept. of Computer Science, Turin, Italy
ruggero.pensa@unito.it

Abstract. In most real world scenarios, experts dispose of limited background knowledge that they can exploit for guiding the analysis process. In this context, semi-supervised clustering can be employed to leverage such knowledge and enable the discovery of clusters that meet the analysts' expectations. To this end, we propose a semi-supervised deep embedding clustering algorithm that exploits triplet constraints as background knowledge within the whole learning process. The latter consists in a two-stage approach where, initially, a low-dimensional data embedding is computed and, successively, cluster assignment is refined via the introduction of an auxiliary target distribution. Our algorithm is evaluated on real-world benchmarks in comparison with state-of-the-art unsupervised and semi-supervised clustering methods. Experimental results highlight the quality of the proposed framework as well as the added value of the new learnt data representation.

1 Introduction

Clustering is by far one of the most popular machine learning task among computer scientists, machine learning specialists and statisticians. Although it is conceived to work in fully unsupervised scenarios, very often, its application in real-world domains is supported by the availability of some, scarce, background knowledge. Unfortunately, producing or extracting such background knowledge (in terms of available class labels or constraints) is a time consuming and expensive task. Hence, the amount of available background knowledge is not sufficient for driving a supervised task. Still, it can be helpful in guiding a semi-supervised learning process.

The aim of semi-supervised clustering is to take advantage of the few available side information to guide the clustering process towards a partitioning that takes into account both the natural distribution of the data and the expectations of the domain experts. One of the most popular class of semi-supervised clustering algorithms exploit the so-called pairwise constraints: the clustering process is driven by a set of *must-link* (or similarity) and *cannot-link* (or dissimilarity) pairs modeling the fact that two data examples involved in any of these constraints should belong to the same cluster (must-link) or not (cannot-link). Such

constraints are successively exploited to either learning a distance metric [14, 7, 12] or forcing constraints during the clustering process [23], although the most effective methods usually combine both strategies [4, 3, 19].

However, all these strategies suffer from the same two problems: i) two examples involved in a cannot-link constraint may actually be assigned to the wrong clusters and still satisfy the constraint; ii) when constraints are generated from the labeled portion of the training set (a common practice in semi-supervised learning), and the class is rather loose (e.g., multiple clusters co-exist within the same class), the must-link constraints would mislead the clustering algorithm resulting in poor partitioning results. To address this issue, an alternative form of supervision has been proposed: given three data examples x_a, x_p and x_n, one may impose that x_a (called reference or anchor example) is closer to x_p (called positive example) than to x_n (called negative example). Such relative comparisons form the so-called triplet constraints [15].

In this paper, we propose $Ts2DEC$ (Triplet Semi-Supervised Deep Embedding Clustering). $Ts2DEC$ is a deep embedding-based clustering framework that leverages triplet constraints to inject supervision in the learning process. The framework consists of a two-stage approach: i) an autoencoder extracts a low-dimensional representation (embedding) of the original data and ii) an initial cluster assignment is refined via the introduction of an auxiliary target distribution [25]. Both stages are guided by the knowledge supplied by triplet constraints.

By means of an extensive experimental study conducted on several real-world datasets, we show that our approach outperforms state-of-the-art semi-supervised clustering methods no matter how much supervision is considered.

2 Related Work

Early semi-supervised approaches used pairwise (e.g., must-link and cannot-link) constraints to learn a metric space before applying standard clustering [14], or to drive the clustering process directly [23]. In [23], a simple adaptation of k-means that enforces must-link and cannot-link constraints during the clustering process is described. [2] proposes a constrained clustering approach that leverages labeled data during the initialization and clustering steps. Instead, [4] integrates both constraint-based and metric-based approaches in a k-means-like algorithm. Davis *et al.*, propose an information-theoretic approach to learning a Mahalanobis distance function [7]. They leverage a Bregman optimization algorithm [1] to minimize the differential relative entropy between two multivariate Gaussians under constraints on the distance function. This approach has been recently extended by Nogueira *et al.*, who combine distance metric learning and cluster-level constraints [19]. Zhu *et al.* present a pairwise similarity framework to perform an effective constraint diffusion handling noisy constraints as well [28].

In recent years, the advances in the deep learning field have also fostered new research in semi-supervised clustering. For instance, in [11], the author propose a semi-supervised clustering algorithm that directly exploits labels, instead of pairwise constraints. Their algorithm generates an ensemble of multiresolu-

tion semi-supervised autoencoders. The final partitioning is obtained by applying k-means on the new data representation obtained by stacking together all the different low-dimensional embeddings. In a very recent work [21], a semi-supervised extension to Deep Embedded Clustering (DEC [25]) is proposed. DEC learns a low-dimensional representation via autoencoder and, successively, it gradually refines clusters with an auxiliary target distribution derived from the current softcluster assignment. Its semi-supervised extension [21], called Semi-supervised Deep Embedded Clustering (SDEC) makes use of pairwise constraints in the cluster refinement stage. Therefore, the learned feature space is such that examples involved in a must-link (resp. cannot-link) constraint are forced to be close (resp. far away) from each other.

Our approach is also based on DEC, but, contrary to [21], it exploits triplet constraints introducing the background knowledge at the different stages of the process: during the embeddings generation and during the clustering refinement. We remind that, the expressiveness of triplet constraints has already demonstrated to be effective in the constrained clustering task [15].

3 Triplet Semi-Supervised Deep Embedding Clustering

In this section, we introduce our semi-supervised clustering approach, called $Ts2DEC$ (Triplet semi-supervised Deep Embedding Clustering). The goal is to group together a set of examples $X = \{x_i\}_{i=1}^{N}$ into C clusters given some background knowledge in terms of constraints.

To this purpose, we model our problem using neural networks. In a nutshell, given $ML = \{(x_j, x_l)\}$ (resp. $CL = \{(x_j, x_l)\}$) the set of must-link (resp. cannot-link) constraints, first we derive triplet constraints from these two sets. A triplet constraint is defined as a tuple (x_a, x_p, x_n) where x_a is the anchor example and x_p (resp. x_n) is the positive (resp. negative) example with the associated semantic that x_a and x_p (resp. x_a and x_n) belong (resp. do not belong) to the same cluster. Furthermore, due to transitivity, we also have that x_p and x_n do not belong to the same cluster. Successively, due to the exponential number of triplets we can generate, we adopt a simple and practical strategy to sample a subset of such triplets. We remind that triplet selection is an hard task and some research works are investigating how to smartly sample useful and informative subsets of triplet constraints [26]. It is out of the scope of this work supplying a method that competes with such strategies. On the other hand, we set up an easy and ready to use approach that well fits our scenario. Once the set of triplet constraints are chosen, we inject such background information into a deep-learning based clustering algorithm [26, 9, 27].

More in detail, we integrate the semi-supervision during: i) the data embedding generation, by alternating unsupervised and semi-supervised optimization of the network parameters and ii) the clustering refinement stage when cluster assignment hardening loss [18] is employed. Figure 1(a) and Figure 1(b) provide a general overview of the embedding generation and clustering refinement stage, respectively. For each stage, we depict with the rose color and the dotted line

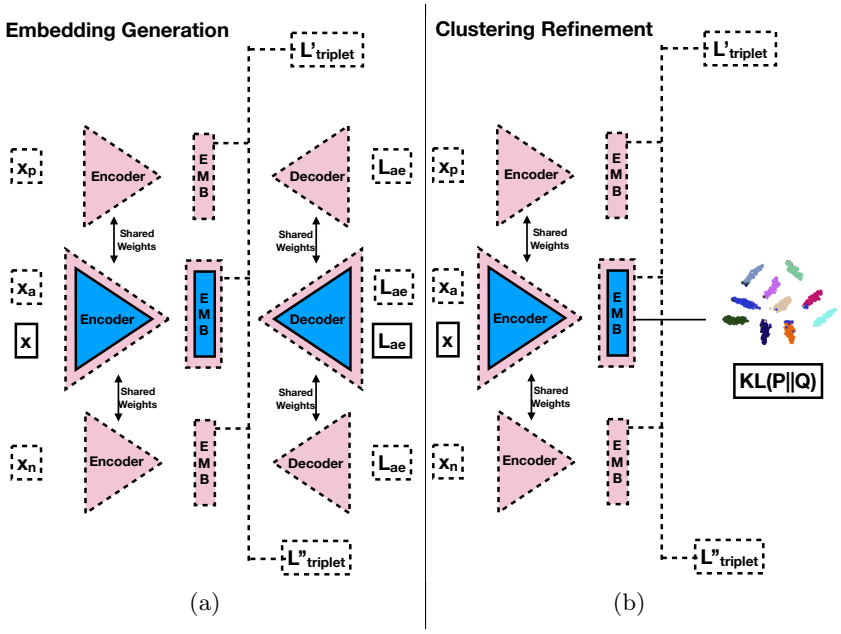


Fig. 1. General Overview of $Ts2DEC$: (a) Embedding Generation and (b) Clustering Refinement. We depict with the rose color and the dotted line the components related to the semi-supervised optimization (working on triplet constraints) while we depict with the blue color and the solid line the fully unsupervised components (working on the whole set of data X).

the components related to the semi-supervised optimization (working on triplet constraints) while we depict with the blue color and the solid line the fully unsupervised components (working on the whole set of data X). In the following, we provide the details of all the algorithmic steps of our approach.

Triplets generation strategy The first preliminary step of $Ts2DEC$ is the generation of a set T of triplet constraints from the set ML and CL of must-link and cannot-link constraints. To achieve this goal, first, we compute the transitive closure from both sets [6], then, we leverage it to generate all possible triplet constraints (x_a, x_p, x_n). Generating triplets in such a way can produce a huge number of constraints. Consequently, we limit the number of triplet constraints by adopting the following strategy: for each pair (x_a, x_p), we randomly sample a subset of possible examples that can play the role of negative examples (x_n). Here, we give more importance to background knowledge that groups together similar examples (positive information) than information that forces examples to be clustered apart (negative information). We adopt this strategy because, during our experimental evaluations, we have empirically observed that positive information seems more effective in stretching the representation manifold thus

respecting the given background knowledge. In our experiments, we sample 30% of all possible negative examples for each pair (x_a, x_p) in order to obtain a reasonable trade off between performances and computational cost. The obtained set of triplet constraints is denoted by T.

Embedding generation with background knowledge The core of $Ts2DEC$ involves a first stage in which semi-supervised embedding representations are generated by means of autoencoder neural networks. This stage is depicted in Figure 1(a). Autoencoders [16] are a particular kind of feed-forward neural network commonly employed to generate low-dimensional representation of the original data by setting up a reconstruction task. The autoencoder network is composed by two parts: i) an encoder network that transforms the original data X into an embedding representation (EMB) and ii) a decoder network that reconstructs the original data from the embedding representation. Furthermore, the autoencoder network is layered and symmetric and the last layer of the encoder part is generally referred as bottleneck layer. The commonly adopted loss function optimized by an autoencoder network is the mean squared error between the original data and the reconstructed one:

$$L_{ae} = \frac{1}{|X|} \sum_{x_i \in X} ||x_i - dec(enc(x_i, \Theta_1), \Theta_2)||_2^2 \tag{1}$$

where $enc(z, \Theta_1)$ is the encoder network with parameters Θ_1, while $dec(\cdot, \Theta_2)$ is the decoder network that reconstructs the data, with parameters Θ_2.

For the encoder network, similarly to what proposed in [25], we adopt a feed-forward neural network with four layers (resp. 500, 500, 2000, 10 neurons per layer). The activation function associated to the first three (hidden) layers is the Rectifier Linear Unit (ReLU) while, for the last (bottleneck) layer, a simple linear activation function is employed [25]. The decoder is symmetrically derived from the encoder reversing the hidden layers.

A semi-supervised autoencoder [8, 20] (denoted as SSAE), instead, is a multi-task network that, in addition to the reconstruction task via its autoencoder structure, also deals with a discrimination task (mainly classification) leveraging the embedded representation. Conversely to most previous works on semi-supervised autoencoders [8, 20, 11] where the SSAE exploits labeled data to perform classification as supervised task, here, we design a SSAE that, associated to the reconstruction task, exploits the set T of triplet constraints to generate the low-dimensional data embeddings EMB.

The set T of triplets is used to learn a triplet network which consists of three different encoders/decoders with shared weights (highlighted in rose color and dotted line in Figure 1(a)). In addition to the standard reconstruction loss, the specific loss function (triplet loss) optimized by the model is defined as follows:

$$L^{'}_{triplet} = \sum_{(x_a, x_p, x_n) \in T} [d(x_a, x_p) - d(x_a, x_n) + \alpha]_+ \tag{2}$$

with

$$d(b,c) = ||norm_{L_2}(enc(b,\Theta_1)) - norm_{L_2}(enc(c,\Theta_1))||_2^2 \quad (3)$$

where T is the set of triplet constraints, $[x]_+ = max(0,x)$ is the hinge loss, $||x||_2^2$ is the squared L_2 norm of x, $enc(x,\Theta_1)$ is the encoder network, with weights parameters Θ_1, applied on an example x, $norm_{L_2}$ is a function that performs the L_2 normalization of the output of the encoder and α is the margin hyperparameter usually involved in distance-based loss function to stretch the representation space [26]. We consider α equal to 1.0 since distances are derived by L_2 normalization.

Additionally, we can observe that, due to the transitivity relation among the examples in the triplet tuple, we can also define a second triplet loss function:

$$L_{triplet}^{''} = \sum_{(x_a,x_p,x_n)\in T} [d(x_a,x_p) - d(x_p,x_n) + \alpha]_+ \quad (4)$$

where the second term of the hinge loss, this time, consider the relationship between the x_p and x_n examples. In the rest of the paper, $L_{triplet}^{'}$ and $L_{triplet}^{''}$ are exploited to introduce semi-supervision in the clustering process and we use the notation $L_{triplet}$ to indicate the sum of the two triplet loss functions: $L_{triplet} = (L_{triplet}^{'} + L_{triplet}^{''})$.

The overall architecture of our semi-supervised autoencoder involves the optimization of $L_{triplet}$ loss as well as the simultaneous reconstruction of the examples concerned by the constraints. Given T, the set of triplet constraints, the loss function optimized by the $SSAE$ is as follows:

$$L_{ssae} = \frac{1}{|T|}\left(\left[\sum_{t\in T}\sum_{x_i\in t} ||x_i - dec(enc(x_i,\Theta_1),\Theta_2)||_2^2\right] + \lambda L_{triplet}\right) \quad (5)$$

where $t = (x_a, x_p, x_n)$ is a generic triplet, λ is a hyperparameter that controls the importance of the triplet loss term. Such loss function optimizes the parameters Θ_1 and Θ_2 so as to optimize the data reconstruction as well as to meet the constraint relationships expressed by the background knowledge. In L_{ssae}, the reconstruction term is considered with the aim of regularizing the action of the $L_{triplet}$ loss. Therefore, we obtain embeddings that meet the requirements expressed by the constraints as well as with the main reconstruction task.

We underline that, in our context, the embedding generation process involves two different stages: the first one implies the optimization of the autoencoder loss on the full set of data X while, the second one regards the optimization of the semi-supervised autoencoder loss considering only the set of examples X_t ($X_t = \{x_i \in t | t \in T\}$) covered by the triplet constraint set. Algorithm 1 reports the joint optimization procedure we employ to learn the weight parameters Θ_1, Θ_2. In a generic epoch, the procedure optimizes: i) the unsupervised loss associated to data reconstruction on the set of data X (line 3-4) and, ii) both reconstruction and triplet losses (L_{ssae}) considering the set of data involved in the set T (line 5-6). The learning of parameters is achieved via a gradient descent based approach using mini-batches. Finally, the data embeddings are generated considering the Θ_1 parameters associated to the encoder network $enc(\cdot,\Theta_1)$.

Algorithm 1 Semi-supervised autoencoder optimization

Require: X, T, N_EPOCHS
Ensure: Θ_1, Θ_2.
1: i = 0
2: **while** i < N_EPOCHS **do**
3: Update Θ_1 and Θ_2 by descending the gradient:
4: $\nabla_{\Theta_1,\Theta_2} \frac{1}{|X|} \sum_{x_i \in X} ||x_i - dec(enc(x_i, \Theta_1), \Theta_2)||_2^2$
5: Update Θ_1, Θ_2 by descending the gradient:
6: $\nabla_{\Theta_1,\Theta_2} \frac{1}{|T|} \left(\left[\sum_{t \in T} \sum_{x_i \in t} ||x_i - dec(enc(x_i, \Theta_1), \Theta_2)||_2^2 \right] + \lambda L_{triplet}(T) \right)$
7: i = i + 1
8: **end while**
9: **return** Θ_1, Θ_2

Clustering refinement with background knowledge Once the embedding representation produced by the SSAE is obtained, the final stage consists in a clustering refinement step via cluster assignment hardening [18, 25] as depicted in Figure 1(b). Here, we iterate between computing an auxiliary target distribution and minimizing the Kullback-Leibler (KL) divergence with respect to it. More in detail, as depicted in Figure 1(b), we discard the decoder part of the previous model (Θ_2 parameters) but we still allow modifications of encoder parameters Θ_1. Given the initial cluster centroids $\{c_j\}_{j=1}^{|C|}$, the cluster assignment hardening technique tries to improve the partitioning using an unsupervised algorithm that alternates between two steps: i) compute a soft assignment between the embeddings and the cluster centroids and ii) update the embedded data representation and refine the cluster centroids by learning from current high confidence assignments leveraging an auxiliary target distribution. The process is repeated until convergence is achieved or a certain number of iterations is executed. To generate the clustering centroids we use K-Means on the embeddings produced by the encoder network.

To compute the soft assignment, as commonly done in deep embedding clustering approaches, we exploit the Student's t-distribution as a kernel to measure the similarity [17]:

$$q_{ij} = \frac{(1 + ||EMB_i - c_j||^2)^{-1}}{\sum_{l=1}^{|C|}(1 + ||EMB_i - c_l||^2)^{-1}} \tag{6}$$

where EMB_i is the embedded representation of the $i-th$ example obtained via $enc(x_i, \Theta_1)$, c_j (resp. c_l) is the cluster centroid of the $j-th$ (resp. $l-th$) cluster, and q_{ij} is the soft assignment between example x_i and cluster c_j.

Once the soft assignments are computed, they are iteratively refined by learning from their high-confidence assignments with the help of an auxiliary target distribution. The target distribution is defined as:

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_l (q_{il}^2 / \sum_i q_{ij})} \tag{7}$$

Such distribution forces the assignment to have stricter probabilities (closer to 0 or 1) by squaring the original distribution and then normalizing it [18].

To match the soft-assignment q with the auxiliary target distribution p, we employ the Kullback-Leibler (KL) divergence as loss function to evaluate the distance between the two probability distributions. The KL divergence is computed between the soft assignment q_i and the auxiliary distribution p_i: $KL(P||Q) = \sum_i p_i \cdot \log \frac{p_i}{q_i}$.

Furthermore, we integrate the semi-supervision supplied by the background knowledge in this step as well, by adding the information carried out by the triplet constraints to the overall loss function:

$$L_{sscr} = KL(P||Q) + \lambda L_{triplet} \tag{8}$$

The resulting loss function considers the auxiliary target distribution together with the triplet constraints when upgrading the parameters of the encoder (Θ_1). Hence, this last step has also an influence on the way embeddings are computed. As before, λ is an hyperparameter controlling the importance of the triplet loss term and it is the same in the two steps of our framework. To optimize such semi-supervised loss L_{sscr} we adopt a similar strategy to what proposed in Algorithm 1. Finally, once convergence is reached, each example is assigned to the cluster that maximizes its assignment score: $cluster(x_i) = argmax_j q_{ij}$.

4 Experiments

In this section, we assess the effectiveness of $Ts2DEC$ on several real world datasets comparing its behavior w.r.t. competitors. Then, we consider the impact of the different components of $Ts2DEC$ by means of an ablation study. Finally, we provide a visual inspection of the representation learnt by our strategy.

Competitors For the quantitative evaluation, we compare the performances of $Ts2DEC$ with those obtained by different unsupervised and semi-supervised competing algorithms. The former are employed as baselines to understand the gain related to the introduction of weak supervision; the latter consist of fair state-of-the-art competitors that are more closely related to the task at hand. As regards the unsupervised approaches, we consider *K-Means* and DEC [25], a recent deep learning unsupervised clustering approach (the unsupervised clustering algorithm $Ts2DEC$ is built upon).
As semi-supervised clustering algorithms, we consider the following competitors: a semi-supervised variant of *K-Means*, named *MPCKmeans* [4]; a recent constrained spectral clustering method [5], called *Spectral*; two very recent deep learning based methods named *MSAEClust* [11] and *SDEC* [21]. *MPCKmeans* combines metric-learning and pairwise constraint processing to exploit the supplied supervision as much as possible. *Spectral* captures constrained clustering as a generalized eigenvalue problem via graph Laplacians. [11] employs an ensemble of semi-supervised autoencoders to learn embedding representations that

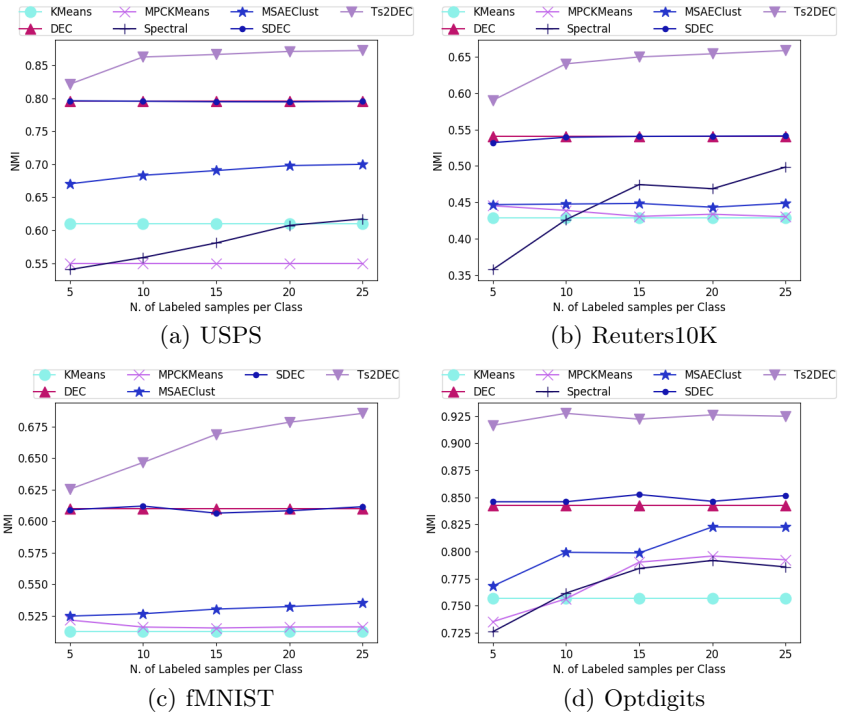


Fig. 2. Results (in terms of NMI) of the different approaches varying the amount of labeled samples per class on: a) *USPS*, b) *Reuters10K*, c) *fMNIST* and d) *Optdigits* benchmarks.

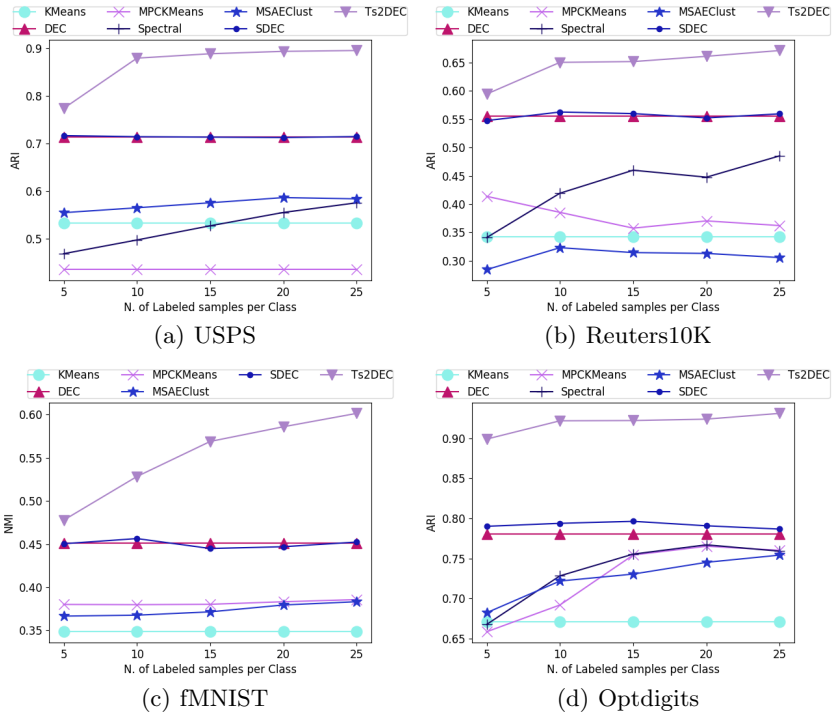
fit the data as well as the background knowledge and that are finally used to perform clustering. Finally, *SDEC* is a direct extension of *DEC* that integrates the pairwise constraints in the clustering refinement stage, by adding an extra term to the cluster assignment hardening loss.

Experimental settings and datasets To measure the clustering performances of all the methods, we use the Normalized Mutual Information (NMI) [22] as well the Adjusted Rand Index (ARI) [10]. Both NMI and ARI take their maximum value when the clustering partition completely matches the original one, i.e., the partition induced by the available class labels. The NMI measure ranges between $[0, 1]$ while the ARI index ranges between $[-1, 1]$. Both evaluation metrics can be considered as an indicator of the purity of the clustering result. For each dataset, both measures are computed considering the whole set of examples, including the ones on which the constraints are defined. We analyze the behavior of the different methods according to increasing levels of supervision. More in detail, we simulate the supervision in term of constraints, by selecting a number of labeled examples per class and, successively, inducing the corre-

sponding full set of constraints. We vary such amount of labeled examples per class between 5 and 25 with a step of 5. Due to the randomness of the sample selection process and the non deterministic nature of the clustering algorithms, we repeat the sample selection step 5 times for each number of per-class labels and, successively, we repeat the clustering process 10 times. For $Ts2DEC$ we derive the corresponding triplet constraints as explained in Section 3. Finally, for each level of supervision, we report the average values of NMI and ARI. For all the methods, the number of clusters is equal to the number of classes.
$Ts2DEC$ is implemented via the *Tensorflow* python library and the implementation is available online [3]. Model parameters are learnt using the Adam optimizer [13] with a learning rate equal to 1×10^{-3} for the autoencoder (reconstruction and triplet loss functions) and we use Stochastic Gradient Descent with learning rate equal to 1×10^{-2} for the Clustering Refinement stage (KL loss function) as done in *DEC* [25] and *SDEC* [21]. We set the value of λ equal to 1×10^{-3}, a batch size of 256 and a number of epochs equal to 50 for the semi-supervised autoencoder. For the refinement clustering stage, we iterate the procedure for 20 000 batch iterations as done in *DEC* [25] and *SDEC* [21]. For all the competitors, we use publicly available implementations. For *SDEC*, the source code was kindly provided by the authors of the related paper. Experiments are carried out on a workstation equipped with an Intel(R) Xeon(R) W-2133, 3.6Ghz CPU, with 64Gb of RAM and one GTX1080 Ti GPU. To evaluate the behavior of all the competing approaches the experiments are performed on four publicly available datasets: (1) *USPS* is a handwritten digit recognition benchmark (10 classes) containing 9 298 grayscale images with size 16 x 16 pixels and provided by the United States Postal Service. (2) *fMNIST* is a dataset of Zalando's article images (shirt, sneakers, bags, etc..) consisting of 70 000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. It serves as a more complex drop-in replacement for the original MNIST benchmark [24]. (3)*Reuters10k* is an archive of English news stories labeled with a category tree that contains 810 000 textual documents. Following [25], we used 4 root categories: corporate/industrial, government/social, markets and economics as labels and excluded all documents with multiple labels. We randomly sampled a subset of 10 000 examples and computed TF-IDF features on the 2 000 most frequent words. (4) *Optdigits* is a dataset of the UCI repository involving optical recognition of handwritten digits. It contains 5 620 examples described by 64 feature each.

Quantitative evaluation Figure 2 and 3 report the performances of the different approaches on the four benchmarks in terms of NMI and ARI, respectively. Notice that *Spectral* was not able to process the *fMNIST* benchmark due to the fact that the original implementation cannot handle a dataset with 70 000 examples. We observe that both NMI and ARI depict a similar situation. At first look, we note that $Ts2DEC$ outperforms all the competing approaches regarding any amount of supervision for all the four benchmarks. In addition, the graphs

[3] `https://gitlab.irstea.fr/dino.ienco/ts2dec`



(a) USPS (b) Reuters10K

(c) fMNIST (d) Optdigits

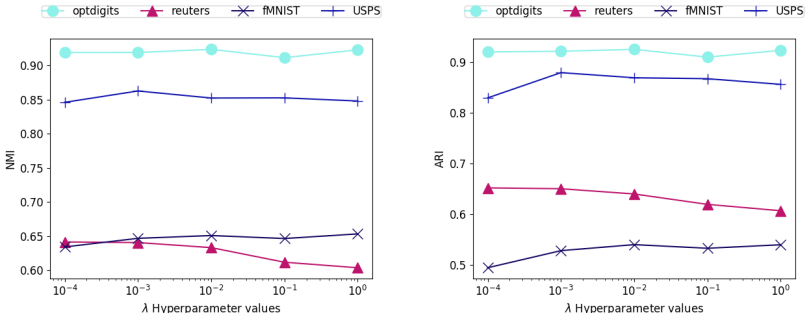
Fig. 3. Results (in terms of ARI) of the different approaches varying the amount of labeled samples per class on: a) *USPS*, b) *Reuters10K*, c) *fMNIST* and d) *Optdigits* benchmarks.

generally show that the margin gained by $Ts2DEC$ increases with the amount of available supervision. This behavior is particularly evident in *USPS*, *fMNIST* and *Reuters10k*. Considering *Optdigits*, we observe an improvement between the supervision value 5 and 10 while, later on, $Ts2DEC$ remains stable according to NMI and it slightly increases according to ARI. This is not the case for all the other semi-supervised competitors. For instance, considering the *fMNIST* benchmark, we note that all competitors remain almost stable while varying the amount of supervision, underlying the fact that they are unable to exploit increasing amount of background knowledge properly. Unexpectedly, we observe that one of the best competitor is DEC, which is completely unsupervised. More surprisingly, *SDEC* performs similarly to its unsupervised counterpart.

Ablation and parameter analysis In this section, we study the impact of the different components of $Ts2DEC$ that involve supervision, as well as the sensitivity of our method to hyperparameter λ. To do this, we fix the amount of supervision by considering 10 labeled examples from each class. For the first study, we derive two variants of our method: i) $Ts2DEC_{v1}$ which considers back-

Table 1. Impact of the different components of $Ts2DEC$ considering the NMI measure.

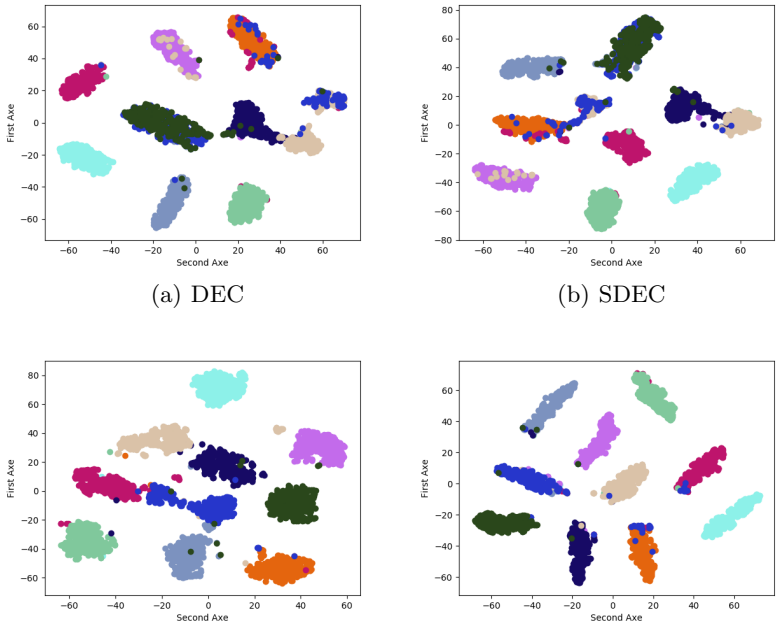
Dataset	$Ts2DEC$	$Ts2DEC_{v1}$	$Ts2DEC_{v2}$
USPS	**0.86** ± 0.02	**0.86** ± 0.02	0.82 ± 0.03
fMNIST	**0.65** ± 0.01	0.64 ± 0.01	0.64 ± 0.02
Reuters10k	**0.64** ± 0.03	**0.64** ± 0.03	0.61 ± 0.03
Optdigits	**0.92** ± 0.01	0.91 ± 0.01	0.91 ± 0.02



(a) NMI (b) ARI

Fig. 4. Sensitivity analysis of the λ hyperparameter: NMI (a) and ARI (b) are reported for increasing weight of semisupervision.

ground knowledge only to generate embeddings via semi-supervised autoencoder, and ii) $Ts2DEC_{v2}$ which considers background knowledge only during the clustering refinement stage. Table 1 reports the results of this study in terms of NMI. We note that the best performances are obtained when semi-supervision is injected at both stages of our process. Furthermore, we observe that $Ts2DEC_{v1}$ consistently achieves slightly better results than $Ts2DEC_{v2}$ in terms of NMI. The results of the sensitivity analysis are given in Figure 4. In details, we let the hyperparameter λ varies in the range $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$. At first look, *USPS*, *fMNIST* and *Optdigits* exhibit a similar behavior. When λ is too small (10^{-4}), supervision is not that effective while, starting from λ equal to 10^{-3}, we observe that $Ts2DEC$ achieves stable performances and becomes insensitive to such parameter. On the other hand, for the *Reuters10k* dataset, we note that the performances slightly decrease when λ increases. At a deeper inspection, we observe that raising the value of λ results in an increase of the standard deviation associated to the average value plotted in Figure 4. We remind that this benchmark is characterized by a high-dimensional feature space (2 000 features) and, the encoder/decoder architecture (inherited from the DEC method) is unable to compress the original data properly and, simultaneously, incorporate the supervision. This may explain the increasing instability and reduced performances when augmenting the importance of the semi-supervision.

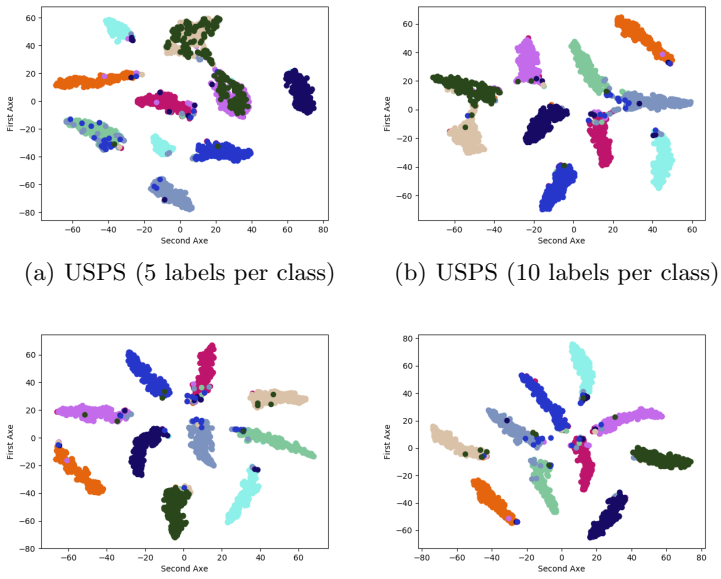


(a) DEC (b) SDEC

(c) MSAECLUST (d) $Ts2DEC$

Fig. 5. Visual inspection of different embeddings processed by *TSNE*.

Visual inspection Here, we visually analyze the embedding generated by our approach on the *Optdigits* benchmark. To this end, we visually compare the embeddings derived by $Ts2DEC$ with the embeddings generated by the other deep learning competitors considering an amount of labeled examples per class equal to 10 (i), and by increasing the amount of background knowledge from 5 to 20 labels per class (ii). To obtain the two dimensional representations, we apply the t-distributed stochastic neighbor embedding (*TSNE*) approach [17]. For this evaluation we consider 300 instances per class. In the former evaluation (Figure 5), we clearly note that the visual representation induced by $Ts2DEC$ provides a better separation among examples belonging to different classes and, simultaneously, locates examples belonging to the same class close to each other. The latter experiment (Figure 5) shows the ability of $Ts2DEC$ to modify the data manifold exploiting the increasing amount of background knowledge. We observe that clear differences exist between the embeddings learnt when 5 (Figure 6(a)) and 15 labeled examples (Figure 6(c)) per class are considered, the latter providing significant better class separation than the former.



(a) USPS (5 labels per class) (b) USPS (10 labels per class)

(c) USPS (15 labels per class) (d) USPS (20 labels per class)

Fig. 6. Visual inspection of the embedding generated by $Ts2DEC$ (and processed by $TSNE$) for increasing amounts of background knowledge.

5 Conclusion

We have presented $Ts2DEC$, a new semi-supervised deep embedding clustering technique that integrates background knowledge as triplet constraints. More precisely, $Ts2DEC$ integrates the background knowledge at two stages: i) during the data embedding generation and ii) during the clustering refinement. Extensive evaluations on real-world benchmarks have shown that $Ts2DEC$ outperforms state-of-the-art competitors w.r.t different amount of background knowledge.

References

1. Banerjee, A., Merugu, S., Dhillon, I.S., Ghosh, J.: Clustering with bregman divergences. Journal of Machine Learning Research **6**, 1705–1749 (2005)
2. Basu, S., Banerjee, A., Mooney, R.J.: Semi-supervised clustering by seeding. In: ICML. pp. 27–34 (2002)
3. Basu, S., Bilenko, M., Mooney, R.J.: A probabilistic framework for semi-supervised clustering. In: KDD. pp. 59–68 (2004)
4. Bilenko, M., Basu, S., Mooney, R.J.: Integrating constraints and metric learning in semi-supervised clustering. In: ICML. pp. 81–88 (2004)
5. Cucuringu, M., Koutis, I., Chawla, S., Miller, G.L., Peng, R.: Simple and scalable constrained clustering: a generalized spectral method. In: AISTATS. pp. 445–454 (2016)

6. Davidson, I., Ravi, S.S.: Intractability and clustering with constraints. In: ICML. pp. 201–208 (2007)
7. Davis, J.V., Kulis, B., Jain, P., Sra, S., Dhillon, I.S.: Information-theoretic metric learning. In: ICML. pp. 209–216 (2007)
8. Haiyan, W., Haomin, Y., Xueming, L., Haijun, R.: Semi-Supervised Autoencoder: A Joint Approach of Representation and Classification. In: CICN. pp. 1424–1430 (2015)
9. Harwood, B., G, V.K.B., Carneiro, G., Reid, I.D., Drummond, T.: Smart mining for deep metric learning. In: ICCV. pp. 2840–2848 (2017)
10. Hubert, L., Arabie, P.: Comparing partitions. Journal of Classification **2**(1), 193–218 (1985)
11. Ienco, D., Pensa, R.G.: Semi-supervised clustering with multiresolution autoencoders. In: IJCNN. pp. 1–8 (2018)
12. Kalintha, W., Ono, S., Numao, M., Fukui, K.: Kernelized evolutionary distance metric learning for semi-supervised clustering. In: AAAI. pp. 4945–4946 (2017)
13. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014), `http://arxiv.org/abs/1412.6980`
14. Klein, D., Kamvar, S.D., Manning, C.D.: From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In: ICML. pp. 307–314 (2002)
15. Kumar, N., Kummamuru, K.: Semisupervised clustering with metric learning using relative comparisons. IEEE Trans. Knowl. Data Eng. **20**(4), 496–503 (2008)
16. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436 (may 2015), `https://doi.org/10.1038/nature14539`
17. van der Maaten, L., Hinton, G.E.: Visualizing high-dimensional data using t-sne. JMLR **9**, 2579–2605 (2008)
18. Min, E., Guo, X., Liu, Q., Zhang, G., Cui, J., Long, J.: A survey of clustering with deep learning: From the perspective of network architecture. IEEE Access **6**, 39501–39514 (2018)
19. Nogueira, B.M., Tomas, Y.K.B., Marcacini, R.M.: Integrating distance metric learning and cluster-level constraints in semi-supervised clustering. In: IJCNN. pp. 4118–4125 (2017)
20. Rasmus, A., Berglund, M., Honkala, M., Valpola, H., Raiko, T.: Semi-supervised learning with ladder networks. In: NIPS. pp. 3546–3554 (2015)
21. Ren, Y., Hu, K., Dai, X., Pan, L., Hoi, S.C.H., Xu, Z.: Semi-supervised deep embedded clustering. Neurocomputing **325**, 121–130 (2019)
22. Strehl, A., Ghosh, J.: Cluster ensembles — A knowledge reuse framework for combining multiple partitions. Journal of Machine Learning Research **3**, 583–617 (2002)
23. Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S.: Constrained k-means clustering with background knowledge. In: ICML. pp. 577–584 (2001)
24. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. CoRR **abs/1708.07747** (2017)
25. Xie, J., Girshick, R.B., Farhadi, A.: Unsupervised deep embedding for clustering analysis. In: ICML. pp. 478–487 (2016)
26. Yu, B., Liu, T., Gong, M., Ding, C., Tao, D.: Correcting the triplet selection bias for triplet loss. In: ECCV. pp. 71–86 (2018)
27. Zhao, Y., Jin, Z., Qi, G., Lu, H., Hua, X.: An adversarial approach to hard triplet generation. In: ECCV. pp. 508–524 (2018)
28. Zhu, X., Loy, C.C., Gong, S.: Constrained clustering with imperfect oracles. IEEE Trans. Neural Netw. Learning Syst. **27**(6), 1345–1357 (2016)