TU Delft

Interactive manipulation of t-SNE embeddings

MASTER'S THESIS

Author
Emiel Bos

Thesis advisor
Prof. dr. Elmar Eisemann

Daily supervisor
Mark van de Ruit, MSc

August 2024

Abstract

Low-dimensional datasets, for which each datapoint contains no more than three attributes, are straightforward to visualize with common visualization idioms, such as scatterplots. In order to visualize high-dimensional datasets with potentially thousands of attributes, their dimensionality will need to be reduced. t-SNE is a widely used, state-of-the-art algorithm for non-linear dimensionality reduction. It produces embeddings of the high-dimensional data onto two or three dimensions by using gradient descent to minimize the discrepancy between the probability distributions of the high-dimensional and low-dimensional datapoint similarities, iteratively adjusting the low-dimensional embedding. However, since it is impossible to capture all neighbourhoods over all high-dimensional axes in two or three dimensions, the resulting clustering is merely virtual, only meant to offer an intuition of the dataset's high-dimensional distribution. Our work proposes a number of tools that allow a user to interactively impose constraints based on prior knowledge or assumptions about the dataset and manipulate the underlying probability model with the aim of enhancing an embedding's interpretability, for a better intuition about the data's high-dimensional structure. We demonstrate the effectiveness and limitations of these tools with examples and case studies.

Contents

1 Introduction

Data visualization (or visual analytics) is the field of data science that studies the visual representation of datasets. Tabular datasets with two attributes can be directly visualized using classical idioms like scatterplots, where the datapoints (the rows) are plotted as points in a two-dimensional[1] graph spanned by the two dimensions (the columns). Datasets with more than three attributes are often more difficult to visually analyze. For example, you cannot directly visualize a dataset of houses with their value, latitude, longitude, and surface area with a two-dimensional scatterplot as is. Nowadays, datasets with millions of datapoints, each with hundreds or thousands of dimensions, are ubiquitous. To reduce the number of dimensions, *dimensionality reduction* techniques can be used. In order to visualize the high-dimensional data with graphs, these techniques reduce the number of dimensions to two or three. The set of datapoints in the reduced space is called a *projection* or *embedding* of the high-dimensional datapoints in the low-dimensional space.

There are two types of dimensionality reduction: linear and non-linear. Linear techniques, such as PCA [2], discard or linearly merge the high dimensions, which preserves the pairwise distance structure amongst all the data samples at the cost of reducing the signal in the dataset. To scatterplot the illustrative houses dataset, you could assign price to one axis, the sum of latitude and longitude coordinates to the other axis, and ignore the surface area. In contrast, non-linear techniques transform the high-dimensional attributes in a non-linear way, such that the low dimensions themselves have no interpretative meaning and no straightforward relation to the original set of high dimensions anymore. However, because non-linear dimensionality reduction keeps datapoints that are close together in the high-dimensional space close together in the low-dimensional space while still preserving global structures like clusters, it can summarize those high-dimensional relations effectively, making it useful for exploratory data visualization. Concerning the houses dataset, you may get a clustering based on all attributes – e.g., one cluster for small penthouses in an expensive city centre, another for countryside manors in a cheap rural area, etc. – but not along well-defined axes.

t-SNE is a non-linear dimensionality reduction technique [31] that has been adopted in the fields of deep learning [24, 12, 7], bioinformatics [3, 28], neuroscience [17], and immunology [5, 32, 16]. t-SNE is a gradient descent algorithm: starting with a random scatterplot, it iteratively nudges all the datapoints in such a way that an objective function is minimized. However, it is generally impossible to translate all high-dimensional neighbourhoods to two dimensions, and there will be some unavoidable projection error in the scatterplot. In this work, we explore possibilities for user interaction to alter the error manifestation and to ultimately generate more representative low-dimensional projections of data.

Section 2 covers related material and prior work, and mainly discusses the original t-SNE technique and its variations. We explain our methods in Section 3 and demonstrate our techniques in Section 4. We discuss the effectiveness and limitations of these techniques and possibilities for future research in Section 5.

[1] This work focuses on two-dimensional data visualization, but nearly everything discussed also applies to three-dimensional visualizations.

2 Background

There are many non-linear dimensionality reduction techniques for visual analytics [36], of which we will mention only a few. One of the earliest is non-metric MDS [21, 6], which attempts to preserve the rank order of all pairwise distances in its embedding. Roweis and Saul state that both PCA and MDS fail to preserve the structure of the non-linear manifold of certain three-dimensional datasets (e.g., the swiss roll point cloud [18]), and propose LLE, in which each embedding point is a learned linear combination of its neighbors [27]. Later, t-SNE was introduced by van der Maaten and Hinton, on which our work is based and which we discuss further in Section 2.1. An alternative formulation of the dimensionality reduction problem was formulated in 2020 with UMAP, namely as a cross entropy minimization between topological representations [20]. They claim it has a stronger mathematical foundation than t-SNE.

2.1 t-SNE

t-distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear dimensionality reduction technique proposed by van der Maaten and Hinton in 2008 [31], which extends Hinton and Roweis's earlier SNE method [10]. Using gradient descent, it generates the low dimensional embedding in such a way to preserve the clusters and local structure of the data that occur in the high dimensional space. In short, t-SNE constructs a similarity distribution over the high-dimensional datapoints, and, starting with a random initialization in the low dimensional space, iteratively moves the low-dimensional counterparts – collectively called the *embedding* or *projection* – of the high-dimensional datapoints to reduce the discrepancy between the distributions.

t-SNE uses a similarity metric between datapoints based on the distances in the high-dimensional space. We assume Euclidean distances, though other distance metrics can be used. For every datapoint $\boldsymbol{x}_i$, all other datapoints $\boldsymbol{x}_j$ are weighted according to a Gaussian distribution centered around $\boldsymbol{x}_i$ with standard deviation σ_i. The similarity of datapoint $\boldsymbol{x}_j$ to datapoint $\boldsymbol{x}_i$ is then the conditional probability $p_{j|i}$ that $\boldsymbol{x}_i$ would pick $\boldsymbol{x}_j$ as its nearest neighbour according to $\boldsymbol{x}_i$'s Gaussian distribution. For $i \neq j$, this is calculated as:

$$p_{j|i} = \frac{v_{j|i}}{\sum_{k \neq i} v_{k|i}} \tag{1}$$

which is a normalization of the individual pairwise similarities $v_{j|i}$, each of which is Gaussian with respect to the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$:

$$v_{j|i} = e^{\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma_i^2}} \tag{2}$$

All $p_{i|i}$ are set to 0. This metric is not symmetric, i.e. $p_{j|i} \neq p_{i|j}$.

The variances σ_i^2 of the Gaussian distributions (should) differ between the datapoints; the variance of a distribution centered around a datapoint in a crowded region of space should be smaller than the variance of a distribution corresponding to a datapoint in a sparse region, because it is

unlikely that one fixed variance for all datapoints would be optimal. This is achieved with the *perplexity* hyperparameter μ, which roughly corresponds to the number of a datapoint's neighbours between which t-SNE will attempt to preserve neighbourhoods.[1] Increasing the variance σ_i for any datapoint $\boldsymbol{x}_i$ increases the Shannon entropy $H(P_i) = \sum_j p_{j|i} \log p_{j|i}$ of that datapoint's distribution P_i. t-SNE performs a binary search for each σ_i by satisfying:

$$2^{H(P_i)} = \mu \tag{3}$$

In order to optimize the gradient computation, van der Maaten and Hinton symmetrize the similarity metric (such that $p_{ij} = p_{ji}$) and further normalize it over all similarity values:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{4}$$

which is a joint probability that intuitively quantifies the probability that two instances $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ would pick each other as nearest neighbours.[2] The joint probabilities p_{ij} between every pair of datapoints are collected in a $n \times n$ similarity matrix P, which is therefore a symmetric joint probability distribution.

Analogous to p_{ij}, we define q_{ij} between embedding datapoints $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$, which are the representations of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in the low-dimensional space; the embedding positions t-SNE seeks to obtain. However, instead of a Gaussian distribution, the Student's t-distribution with one degree of freedom is used for calculating q_{ij}, which places more weight on far neighbours relative to nearby neighbours compared to the Gaussian distribution. The similarity metric for the low-dimensional space becomes:

$$q_{ij} = \frac{w_{ij}}{\sum_k \sum_{l \neq k} w_{kl}} \tag{5}$$

where w_{ij} is the similarity between $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ in the embedding space, though now using the t-distribution rather than a Gaussian distribution:

$$w_{ij} = (1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2)^{-1} \tag{6}$$

The (non-symmetric) Kullback-Leibler (KL) divergence [13, 14, 35] is used as objective (or cost) function:

$$\mathrm{KL}(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{7}$$

[1] It can be thought of as the continuous analogue to k in k nearest neighbour techniques. Another way of looking at perplexity is as a hyperparameter that balances t-SNE's attention between local and global aspects of the data [34]. It is normally set between 5 and 50.

[2] SNE instead uses the $p_{j|i}$ directly in its objective function, which is slower.

where matrix Q again holds all the q_{ij}. $\mathrm{KL}(P||Q)$ is 0 if all p_{ij} are equal to their q_{ij}, i.e., Q would perfectly model P. From this formula, we can derive the gradient for each datapoint; the direction in which to move it:

$$\frac{\partial \mathrm{KL}(P||Q)}{\partial \boldsymbol{y}_i} = 4(F_i^{\text{attractive}} - F_i^{\text{repulsive}}) \tag{8}$$

$$F_i^{\text{attractive}} = \sum_{j \neq i} p_{ij} q_{ij} Z(\boldsymbol{y}_i - \boldsymbol{y}_j) \tag{9}$$

$$F_i^{\text{repulsive}} = \sum_{j \neq i} q_{ij}^2 Z(\boldsymbol{y}_i - \boldsymbol{y}_j) \tag{10}$$

where $F_i^{\text{attractive}}$ is the total attractive force that all other datapoints exert on $\boldsymbol{y}_i$ and $F_i^{\text{repulsive}}$ is the total repulsive force it undergoes. Intuitively, this gradient can be visualized as the average force created by a set of springs from $\boldsymbol{y}_i$ to every other $\boldsymbol{y}_j$, where each springs exerts a force proportional to its length and its stiffness (the mismatch $p_{ij} - q_{ij}$). Concretely, if two low-dimensional datapoints q_i and q_j are too far apart with respect to their high-dimensional counterparts p_i and p_j, there will be an attractive force in the gradient that pulls them together in the next step. If q_i and q_j are too close together, there will be a repulsive force in the gradient. This makes the objective minimization essentially an n-body simulation [1].
In order to speed up the optimization and to avoid poor local minima, a relatively large momentum term is added to the gradient. Therefore, the actual gradient update step looks as follows:

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\partial \mathrm{KL}(P||Q)}{\partial \mathcal{Y}^{(t-1)}} + \alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)} \right) \tag{11}$$

where $\mathcal{Y}^{(t)}$ is the solution at iteration t, η is the learning rate hyperparameter, $\alpha(t)$ is the momentum at iteration t, and $\alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)} \right)$ is an exponentially decaying sum of previous gradients used to speed up the descent and avoid poor local minima.

The authors mention two optimizations that improve results by modifying the objective function for the first number of iterations. *Early compression* adds an L^2-penalty that penalizes long distances of the datapoints to the origin, forcing the datapoints to stay close together and move through each other. This allows for quickly exploring multiple configurations early on. *Early exaggeration* multiplies all p_{ij} by a prespecified factor (usually 12 [20]). In this case, the property $\sum_i \sum_{j \neq i} p_{ij} = 1$ naturally does not hold anymore, and since all q_{ij} still add up to 1, they are too small to model the corresponding p_{ij}s. The effect is that the clusters that occur naturally in the high-dimensional data become tight and widely separated in the low dimension, and the resulting empty space makes it easier for the clusters to move around and, again, quickly explore good global organizations.

2.2 Subsequent t-SNE adaptations

t-SNE has a $O(n^2)$ time and space complexity, because both P and Q require a normalization term summing over all $n(n-1)$ pairs of datapoints [31], making it intractable on large datasets. In 2014, van der Maaten proposes BH t-SNE, which employs a Barnes-Hut approximation to achieve $O(n \log n)$ runtime [30].
First, rather than calculating the similarity between every pair of datapoints, only similarities $p_{j|i}$ between a datapoint $\boldsymbol{x}_i$ and its k nearest neighbours is calculated, because only the closest neighbouring datapoints will have a meaningful value for p_{ij}. Van der Maaten uses $k = 3\mu$. The set $k\text{NN}_i$ needs to be calculated for every datapoint $\boldsymbol{x}_i$, which is done in $O(\mu n \log n)$ time. During gradient descent, only a datapoint's k neighbours exert attractive forces onto it and are calculated. Our method allows for manipulating the embedding with techniques that edit the sparse matrix P and change the structure of the kNN graph.
Second, van der Maaten improves the time complexity of the repulsive force calculations within $\mathcal{Y}$ from $O(n^2)$ to $O(n \log n)$ as well, by viewing it as an n-body simulation, to be approximated by the Barnes-Hut (BH) algorithm [4]. This algorithm first builds a *quadtree* (or *octree* in case of three dimensions) over the embedding on the datapoints. Each node in the tree stores the center of mass of that cell, which is the average of its datapoints. Using the quadtree, the repulsive force of a group of multiple datapoints (within a quadrant of the quadtree) on a datapoint at hand can be approximated by their center of mass if those other datapoints are sufficiently far away.

In 2020, Pezzotti et al.'s Linear t-SNE (L t-SNE) discards the Barnes-Hut algorithm in favour of a field texture obtained with the GPUs rasterization pipeline, in order to quickly sample it for force calculations [26]. Drawing this field texture is still expensive, and therefore DH t-SNE, by van de Ruit et al., reintroduces the Barnes-Hut algorithm [29], not only to approximate densities and repulsive forces of groups of datapoint within embedding hierarchy clusters, but to use those approximations on multiple texels at once within regions of the field hierarchy whenever cluster-and-region pairs are sufficiently distant. DH t-SNE can therefore be regarded as a hybrid between BH t-SNE and L t-SNE, compounding their ideas.

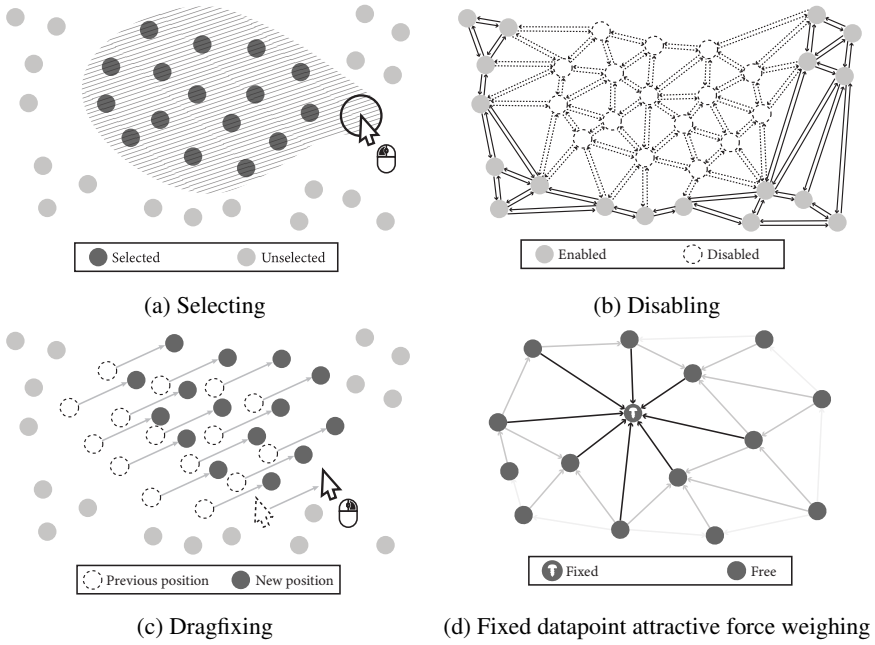
In 2016, before t-SNE adaptations achieved speeds at which they could be used interactively, Pezzotti et al. replaced the prohibitively slow exact computation of kNN sets with approximations and refine this kNN structure during gradient descent within regions selected by brushing, which they call Approximated t-SNE (A t-SNE) [25]. Similarly, the same authors proposed hierarchical stochastic neighbourhood embedding (HSNE) in the same year, which initially offers only a coarse embedding with only the dominant structures, after which the user can "drill down" or "zoom in" to select structures, isolating only those datapoints and offering more detailed visualizations. Both these techniques are examples of the Overview-First, Details-On-Demand philosophy. In 2020, Chatzimparmpas et al. aim to increase interpretability of t-SNE embeddings by visualizing the embedding error [8]. They also provide auxiliary graphs that visualize, among others, neighbourhood preservation metrics for dimensionality reduction methods proposed by Martins [19]. Yet other research efforts focus on integrating constraints into dimensionality reduction in general. Vu et al. present a survey and categorization of constraint types and techniques for various linear and non-linear dimensionality reduction methods [33]. Our work focuses on constraint integration, specifically for t-SNE.

3 Method

t-SNE, and dimensionality reduction in general, inherently introduce errors in embeddings, hindering their interpretability. As a basic example, trying to represent a three-dimensional sphere on a two-dimensional plane necessarily involves cutting the sphere along its surface. Real world datasets tend towards hundreds to thousands of attributes [5, 32, 16, 3], and representing those with two dimensions for exploratory data analysis analogously involves introducing distortions. t-SNE merely attempts to mirror the distribution of high-dimensional distances to the distribution of embedding distances while assuming all attributes weigh equally. Furthermore, t-SNE is unpredictable and sensitive to hyperparameter settings and the dataset at hand, requiring experimentation and finetuning for desirable results [34]. This can result in suboptimal representation quality of generated embeddings. In this section, we propose methods that mitigate this by allowing the user to interactively steer a minimization and the manifestation of these distortions. They work by altering the projected datapoint locations in the embedding, either directly, or indirectly by editing the similarities between datapoints (i.e. the matrix P). In essence, we propose for the user to impose preconceptions about the data and its structure onto t-SNE, as it lacks such information.

Across proposed methods, a fundamental operation is the selection of datapoints. In an interface, we allow users to brush-select embedding datapoints and to edit or invert this selection. Besides a primary selection, a distinct secondary selection can be made, which enables operations such as merging and separation of these selections. Besides the potential presence of class labels, the embedding of a dataset gives no information on what the datapoints represent. To give the user quick intuitions, we summarize the current selection of datapoints *per attribute* in a number of visualizations: average, variance, average difference between neighbouring pairs, average difference between all pairs, and the difference between the last two. These latter auxiliary visualizations are meant to give an understanding of why the algorithm generated the resulting structure. For image datasets, where every datapoint represents an image and the attributes are pixel values, these visualization are images; for other datasets, we simply use barplots. For a given selection, we can also render all neighbours of all selected datapoints as links, with similarities represented as the links' opacity. Datasets may be partially labeled, and for this reason the program offers a mode of selection in which only labeled datapoints can be selected and manipulated.

There are two invariants that (need to) hold at any time: the total sum of similarities in the embedding should be n, and all similarities should be symmetric. If an operation alters similarities and changes their total sum, we perform a renormalization step to bring the sum back to n. If similarities are edited in such a way that there could exist asymmetrical similarities, we perform a symmetrization step that makes all similarities symmetrical again.
The following sections discuss the specific methods. Selected datapoints can be dragged and fixed anywhere in the embedding, explained in Section 3.1. In Section 3.2, we propose a method to merge two erroneously separated subclusters. The inverse operation, cluster separation, is discussed in Section 3.3.



(a) Selecting (b) Disabling

(c) Dragfixing (d) Fixed datapoint attractive force weighing

Figure 1: (a) Datapoint selection by brushing. (b) Datapoint disabling. (c) Selected datapoint dragfixing. (d) Attractive force weight originating from the fixed datapoints and spreading through the embedding. Arrow opacity represents force magnitude.

3.1 Dragfixing datapoints

The embedding space and the axes that span it have no intrinsic or intuitive meaning; t-SNE arbitrarily occupies the space in a way that optimally minimizes the objective function. The user can drag the datapoints in the primary selection across the embedding, as illustrated in Figure 1c. Upon release, the datapoints will be fixed in place, with the rest of the embedding ideally adjusting accordingly. In this way, the user can assign certain (characteristics of) data to certain regions. For example, the thickness of shapes can be distributed across the horizontal axis while the roundness can be distributed across the vertical axis by placing exemplary selections of datapoints at the axes' extremes. The user may also select (part) of a cluster/class and drag it to a desired location (one that may be intuitive based on the attributes of the selection), with the rest of the cluster/class being pulled towards the selection. As an example of this, in a word2vec dataset [22], the user may want to place a cluster of adjectives to the left of a cluster of nouns, as that is how the two types of word often appear in sentences.

Rigidness is a hyperparameter that determines the reduction of the learning rate of fixed points, and ranges between 1 – which completely fixes datapoints – and 0 – at which "fixed" points move at the same velocity as regular datapoints. Dragfixing datapoints is especially useful when used on partially labeled datasets. In the labeled-only selection mode, these labelled datapoints will be visually marked and will be the only selectable datapoints, so that the user can easily select and dragfix them, with datapoints from the same class (or otherwise similar) ideally gravitating toward the labelled datapoint(s).

Without altering the attractive forces, dragging and fixing a small number of datapoints has a negligible effect on the embedding, because their combined attractive force is too weak. Therefore, we weigh the attractive forces $F_{ij}^{\text{attractive}}$ (see Equation 9) emanating from fixed datapoints x_j when the selection is small. Moreover, the neighbours of fixed datapoints in turn weigh their attractive forces, albeit by a reduced factor, in order to attract indirectly connected datapoints and avoid rifts caused by large force differences. In other words, fixed datapoints initiate a weighing of attractive forces that spreads throughout the kNN graph, but gets weaker the further it spreads. This weighing process can be controlled by the user with two hyperparameters: the base weight w of fixed datapoints, and a weight falloff τ. The weight falloff τ is a ratio between 0 and 1 with which the weight gets attenuated each time it spreads further. During attractive force computation, a datapoint $\boldsymbol{x}_i$ calculates its attractive force F_{ij} towards any neighbour $\boldsymbol{x}_j$, multiplies that by the weight $\omega(\boldsymbol{x}_j)$ assigned to $\boldsymbol{x}_j$, and sets its own force weight to $\omega(\boldsymbol{x}_j) * \tau$. Figure 1d illustrates the process. When dragfixing a larger selection of datapoints, the weighing of attractive forces described above is undesirable. Therefore, the actual base weight that selected datapoints get is $\max(\frac{w}{|S|}, 1)$, where $|S|$ is the size of the set of selected datapoints.

3.2 Cluster merging

One type of unwanted distortion that t-SNE embeddings may have is the incorrect separation of datapoints that belong to the same class or that should otherwise be spatially colocated. We provide tools to (re)unite subclusters into one whole. A rudimentary method is to make two distinct selections and to multiply the similarities between datapoints from the different selections. This is ineffective when there are few to no preexisting neighbours between the two different selections. For this reason, we propose a method that adds neighbour pairs between datapoints in the primary selection and datapoints in the secondary selection, with similarities large enough to pull the two clusters towards each other. We call this the *fusing* of two clusters or selections. For brevity, we occasionally refer to neighbour pairs as *links*. *External* links/similarities are those between selected datapoints (from any selection) and unselected datapoints. *Intraselection* links/similarities are those that are between datapoints in the same selection. *Interselection* links/similarities are those between datapoints from different selections. The method only adds interselection links.

In order to conserve performance and memory, the number of new neighbour pairs should not be too large, and we therefore restrict the increase of a datapoint's kNN-set size to a maximum of parameter k per fusing operation execution. This means that a kNN-set increases linearly with the number of fusings that its datapoint is subjected to. For any selected datapoint $\boldsymbol{x}_i$, this is done deterministically by dropping out $1 - \frac{k}{\text{size of other selection}}$ of its neighbour candidates (i.e. the datapoints in the other selection).

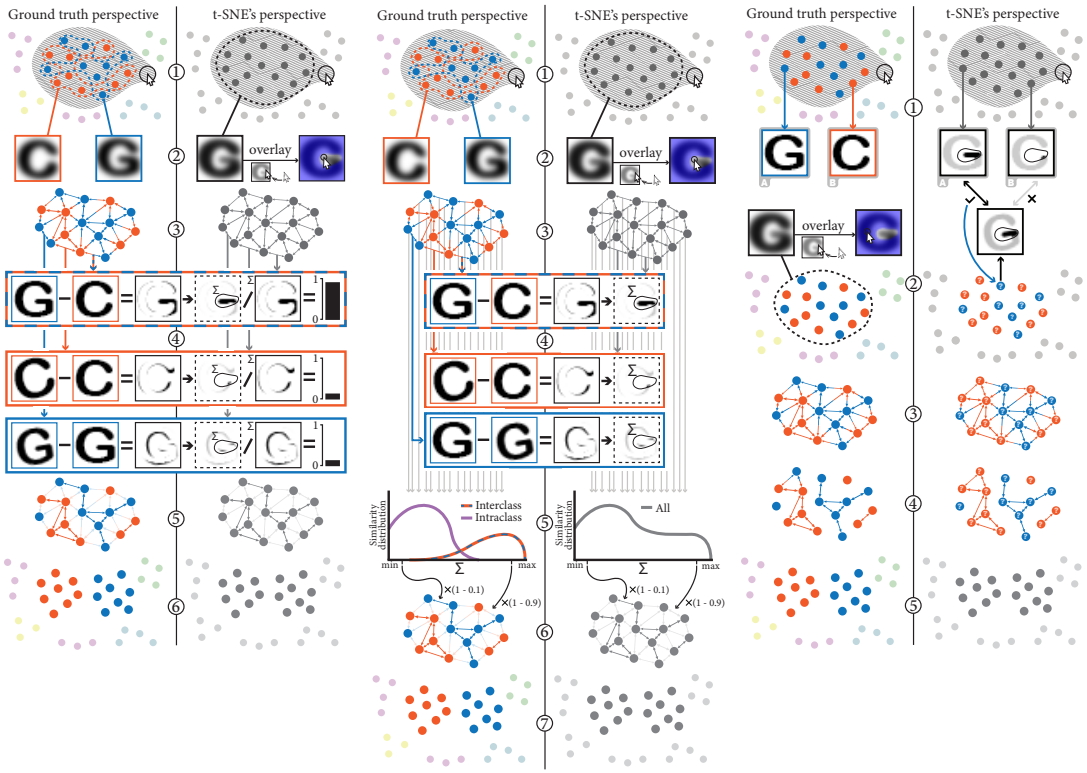
For each selected datapoint $\boldsymbol{x}_i$, we keep the total sum of its similarities equal before and after the operation. This means that a portion of the preexisting similarities is siphoned to the new links. The ratio of this portion can be set by the user; by default, a third of the preexisting similarities goes to the new links, which we found to work well. Secondly, we base the relative values of the added similarities on the Euclidean distances expressed by the datapoint's Gaussian distribution,

just like the preexisting similarities. To this end, we first obtain a new value of σ_i with a binary search that satisfies Equation 3, using only the datapoints at the other ends of its new links for computing $H(P_i)$. With this new σ_i, we can calculate the $p_{j|i}$'s (the asymmetric similarities obtained with Equation 1) and use those as added similarities. These added similarities are calculated exactly as if these new neighbours were $\boldsymbol{x}_i$'s actual k nearest neighbours.

3.3 Cluster separation

As the inverse distortion of t-SNE incorrectly breaking up a cluster into multiple subclusters, it can also erroneously put datapoints from two (or more) distinct classes in the same cluster, or colocate datapoints that are otherwise distinct. In the simple case that two clusters with each one class have simply attached themselves to each other with a clearly visible seam separating the classes, a trivial solution is to simply select the two halves with different selections and remove the interlinks, thereby creating a split where the different selections meet.
It is more challenging to separate a cluster where the datapoints of the different classes have blended together. We propose methods that allow for user input to weight visual features not directly encoded in a dataset. Since t-SNE places equal importance on a dataset's attributes, it may spatially co-locate datapoints from different classes if they have more attributes sharing similar values than attributes on which they differ significantly. We refer to these latter attributes as *discriminatory attributes*, since those attributes can be regarded as holding the information to distinguish between the two classes. For example, in a textual dataset of dish recipes, t-SNE could consider enchiladas and burritos to be similar, because they share the same cuisine and roughly the same ingredients and taste. We offer ways to tell t-SNE to emphasize the discriminatory attributes – in our example, those could be the size/shape of the dish and method of preparation – relative to all other attributes, and to adjust the similarities between selected datapoints accordingly.

We iteratively developed four methods of separation. The first two, ratio-based and range-based separation, require the same user input but differ in how they calculate the similarity reductions. After those proved ineffective in some scenarios, we developed resemblance-based separation, which requires more user input. We lastly developed a method that facilitates separation in a manual, almost "brute-force" manner, for situations in which the three main methods fail. Figure 2 summarizes the three main procedures. All methods require the user to select the cluster to be separated with a single (primary) selection. In any of the attribute visualizations, the user can select what they think are the discriminatory attribute. We use the Manhattan/L1 distance when computing the similarities between datapoints, which is simply the sum of the differences between two datapoints' attribute values. We will refer to this as the *total difference* between two datapoints, in order to differentiate it with the *subdifference*, which sums the difference only over a strict subset of attributes. Furthermore, we distinguish between *interclass* neighbours, pairs of neighbouring datapoints that do not belong to the same class, and *intraclass* neighbours, which do belong to the same class. The methods should cause similarities between interclass neighbours to be reduced more than the similarities between intraclass neighbours. Our examples and demonstrations separate two classes, but our methods support separating more classes as well.



(a) Ratio-based (b) Range-based (c) Resemblance-based

Figure 2: The three methods of cluster separation. The left side of each diagram shows the process from a ground truth perspective (i.e., with datapoint labels), while the right side shows the process from the perspective of t-SNE (i.e., without label information).
(a) 1. Cluster selection. 2. Average attribute values and user-selected attributes. 3. Unicolored arrows represent intraclass similarities and duocolored arrows represent interclass interclass. 4. Calculation of the reduction factor of an interclass similarity (top) and two intraclass similarities (bottom). 5. Similarity reduction. 6. Resulting cluster split.
(b) 1, 2, 3. Idem. 4. Calculation of the subdifference rather than the ratio. 5. Calculation of the reduction factor based on the interval spanned by the minimum and maximum of all subdifferences of all selected pairs. 6. Similarity reduction. 7. Resulting cluster split.
(c) 1. Cluster selection and archetype setting. 2. Average attribute values and user-selected attributes. 3. Similarity either as interclass or intraclass. 4. Interclass similarities are set to zero. 5. Resulting cluster split.

3.3.1 Ratio-based cluster separation

This method uses the subdifference along the selected attributes as a ratio to the total difference to reduce each similarity. The intuition is that the larger the subdifference, the more likely that the two datapoints are from different classes, hence the larger the reduction. Each similarity p_{ij} between selected datapoints $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is updated as follows:

$$p_{ij}^{(\text{new})} = p_{ij}^{(\text{old})} * (1 - \alpha) \tag{12}$$

$$\alpha = \frac{\sum_{a \in A} |\boldsymbol{x}_i^{(a)} - \boldsymbol{x}_j^{(a)}|}{||\boldsymbol{x}_i - \boldsymbol{x}_j||_1} * m \tag{13}$$

where α is the reduction factor, A is the set of selected attributes, $\boldsymbol{x}_i^{(a)}$ is the value of attribute a of datapoint $\boldsymbol{x}_i$, and $||\boldsymbol{x}_i - \boldsymbol{x}_j||_1$ is the total difference between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Intuitively, a part of the similarity equal to the proportion of a's subdifference to the total difference is removed. For interclass similarities, this part is large, which will cause a relatively large reduction, while for intraclass similarities, a's contribution to the total difference is small, which will cause only a minor reduction.

Purely using the ratio of attribute difference to total difference means that the size of the set of selected attributes would determine the magnitude of the similarity reduction, which is undesirable. This also entails that the reduction is small for small sets of selected attributes. We therefore use a multiplier m to increase the effect of the recalculation. The multiplier is based on the number of selected attributes $|A|$ relative to the total dimensionality d:

$$m = \frac{\frac{d}{|A|}}{a * (1 - \frac{|A|}{d}) + 1} \tag{14}$$

The main part of this multiplier is $\frac{d}{|A|}$, which is large when a small number of attributes are selected, and small when many attributes are selected. Using only $\frac{d}{|A|}$ results in exorbitant reduction of intraclass similarities, and therefore we divide by a scalar a, for which we found $a = 5$ to suffice in practice. However, in order to retain the property that weighing all attributes with zero should also reduce similarities to zero, we multiply this division by $1 - \frac{|A|}{d}$.

3.3.2 Range-based cluster separation

The factor in Equation 13 considers each similarity in isolation, and only uses the set of discriminatory attributes as user-supplied information. The second method is similar algorithmically and in user experience, but additionally takes the distribution of subdifferences across all pairs of neighbours in the selection into account, taking as factor the magnitude of the subdifference

relative to the subdifferences of all other selected neighbouring pairs. The subdifferences between datapoints from different classes should be near the larger end of the spectrum, whereas subdifferences betweem datapoints from the same class should be on the smaller end.

For each pair of selected, neighbouring datapoints, the subdifferences along all selected attributes are summed. The interval between the minimum and maximum of all these sums is used as a range, and the relative position of the summed subdifference of every similarity within this range is used to determine its reduction. That is, the reduction factor is now calculated as:

$$\alpha = \frac{\sum_{a \in A} |\boldsymbol{x}_i^{(a)} - \boldsymbol{x}_j^{(a)}| - \min}{\max - \min} \tag{15}$$

where $\min$ and $\max$ are the smallest and largest of all summed subdifferences, respectively. In short, similarities of which the subdifference is on the high end of the range are reduced more than similarities that lie on the low end of the range.

3.3.3 Resemblance-based cluster separation

The third method solicits more information from the user, namely one or more typical examples of each class. We refer to these examples as *archetypes* of a class. The program uses these to classify each similarity either as an interclass or intraclass similarity. This is done by first classifying each datapoint by picking the archetype with the smallest subdifference to that datapoint. If archetypes of different classes are picked, the program classifies it as an interclass similarity, and if the same archetypes are picked, the similarity is classified as intraclass. We discuss this method for separating two classes of datapoints with two archetypes, but our program also allows separating three or more classes, each with any number of archetypes. Although, it is expected that selecting attributes will benefit this method's effectiveness, it is not fundamental, as was the case with the previous two methods.

Archetypes are set from a list of suggestions made by the program based on the current selection. The program uses the k-medoids algorithm, an adaptation of k-means that, rather than yielding centroids (which can be anywhere in the space), yields medoids, which are datapoints. The medoid of a cluster is the datapoint whose average dissimilarity to all other datapoints in the cluster is the smallest. These medoids are then proposed as archetype suggestions. Each suggestion can be set and unset as an archetype of at most one class.

For every selected datapoint $\boldsymbol{x}$, the archetype with the smallest subdifference to it is determined:

$$\operatorname*{argmin}_{r \in \mathcal{A}} \sum_{a \in A} |\boldsymbol{x}^{(a)} - \boldsymbol{x}_r^{(a)}| \tag{16}$$

where $\mathcal{A}$ is the set of archetypes. A similarity is estimated to be an interclass similarity if its two datapoints differ in their closest archetype, while a similarity is estimated to be an intraclass similarity if the same archetype is assigned to its datapoints. In the former case, the similarity is reduced to 0, while in the latter case the similarity remains untouched.

3.3.4 Manual cluster separation

We lastly propose a more simplistic, "brute-force-like" approach. The user can use the previously discussed list of archetype suggestions to assign as many as reasonable. The method then fixes all unselected datapoints, disables all selected datapoints that are not archetypes, and adds links between every pair of archetypes with the same assigned archetype class[1]. The program then runs in "manual separation mode", in which only the archetype datapoints move around. The similarities between archetypes of the same class are doubled, the similarities between archetypes of different class are negated[2], and the similarities between archetypes and regular datapoints are ignored. The use can manually aid this clustering and position the archetypes of this does not happen sufficiently satisfactorily on its own. Exiting the "manual separation mode" then fixes the archetypes, which increases their attractive force towards non-archetype datapoints. Since it is often undesirable to have datapoints fixed in order to separate a cluster, the rigidness of fixed datapoints can be set to zero to only increase their attractive force without fixing them in the embedding. A benefit of this method is that a cluster's separation can be iteratively improved; if chunks of the cluster have not been properly separated, the problematic area can be selected in order to indicate additional archetypes from only this region, after which the process can be retried with the extra information.

3.4 Implementation

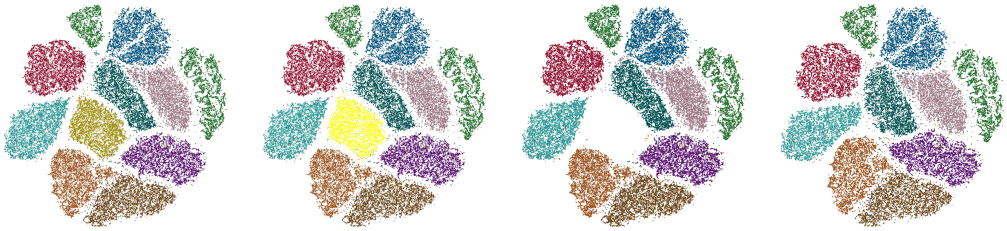
Our project is available as a Git repository hosted on GitHub.[3]. It is a fork of van de Ruit et al.'s dual-hierarchy t-SNE implementation [29]. Due to system limitations, we set the value of θ for the dual-hierarchy to 0, meaning the algorithm degenerates to use only a single hierarchy. This does not affect our findings, since our research is not concerned with performance. Our project is written mainly in C++ and uses OpenGL 4.6 and CUDA 11 for GPGPU, the Faiss library [11] for k nearest neighbour search, k-means clustering and k-medoids search, and Dear ImGui.

We briefly mention two more useful but simple additional operations: disabling and isolating. Disabled datapoints have their visual representation removed from the embedding and their (attractive and repulsive) forces skipped during force calculation. This allows the user to disregard (clusters of) datapoints that are of no interest, freeing up embedding space in the process. They can be reinserted. Our application also allows us to restart the entire algorithm on a subset of the dataset, which we refer to as "isolating", which is useful for focusing on a part of the dataset. This recalculates the kNN graph and similarities, optionally with different options for k and perplexity. In our implementation, unselected datapoints are removed from memory when isolating, making the operation irreversible.

[1] By definition, the archetypes suggested by k-medoids lie relatively far apart in the high-dimensional space, and as a result many archetypes are not neighbours.

[2] This essentially transforms the attractive forces into repulsive forces. We cannot use t-SNE's actual repulsive forces, since those are calculated with a field texture and therefore individual repulsive forces between specific datapoints cannot be easily targeted.

[3] `https://github.com/EmielBoss/dual_hierarchy_tsne`



(a) (b) (c) (d)

Figure 3: (a) t-SNE embedding of MNIST. (b) The highlighted cluster is selected. (c) The selected datapoints are disabled. (d) After a few iterations, t-SNE reclaims the freed space and converges on a new embedding.

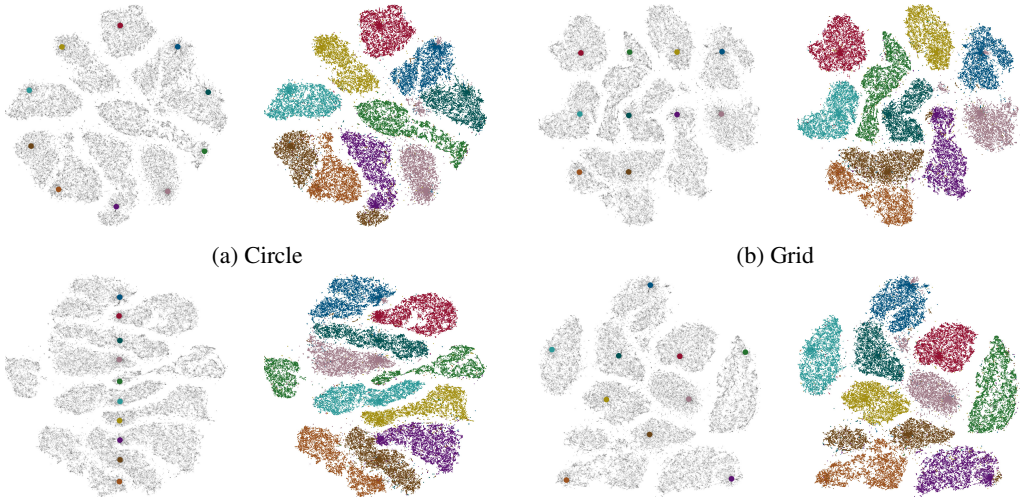
4 Results

In this section, we demonstrate and evaluate operations introduced in the preceding Section 3. Due to nature of the research and our lack of appropriate datasets, all experiments are anecdotal. Figure 3 shows a cluster from an MNIST embedding being selected and disabled. The hole that appears after the cluster is disabled is quickly occupied again by the surrounding datapoints. We briefly discuss the dragfixing of datapoints in Section 4.1. Section 4.2 demonstrates cluster merging and Section 4.3 demonstrates and evaluates all methods of cluster separation. In these sections, we use embeddings of two datasets: MNIST and EMNIST [15, 9]. Both datasets consist of 60,000 28×28 grayscale images. They depict handwritten numbers `0` through `9` in the case of MNIST, and handwritten letters `a`/`A` through `z`/`Z` in the case of EMNIST, where lowercase and uppercase versions of each letter share the same class. Section 4.4 takes a dataset-centric approach with case studies of how two other datasets can be improved using a variety of our proposed techniques.

4.1 Dragfixing datapoints

For our demonstration of the dragging and fixing of datapoints, we took the fully labeled MNIST dataset and pretended to only have one datapoint per class labeled in the labeled-only selection mode. Figure 4 shows how we manipulate the embedding into four different arrangements with only these singular datapoints. In the case of MNIST, which is an easy dataset for t-SNE to cluster, the embedding adapts well. Section 4.4.2 shows the technique on a more difficult dataset. The embedding of that dataset is not nearly as well-clustered, in which case dragfixing is not hardly useful.

Another application of dragfixing is the shaping of individual clusters to control the layout of their datapoints. For example, in the MNIST dataset, the majority of `1`'s are written as one line. The degree of tilt of this line is the human-identifiable property with the most variation among attributes, and as a result the kNN graph structure and the cluster seem to be shaped with respect to this property the most. We can use the selection brush and the attribute visualization to find the datapoints which have their line slanted to the left, and those which have their line slanted to the right. We then drag those opposites to two different locations in the embedding,



(a) Circle (b) Grid

(c) Vertical line (d) Star

Figure 4: Four different arrangements of ten fixed datapoints and the corresponding adaptation of t-SNE's MNIST embedding

.

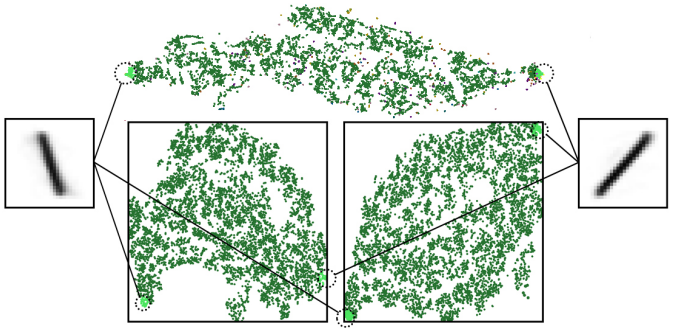
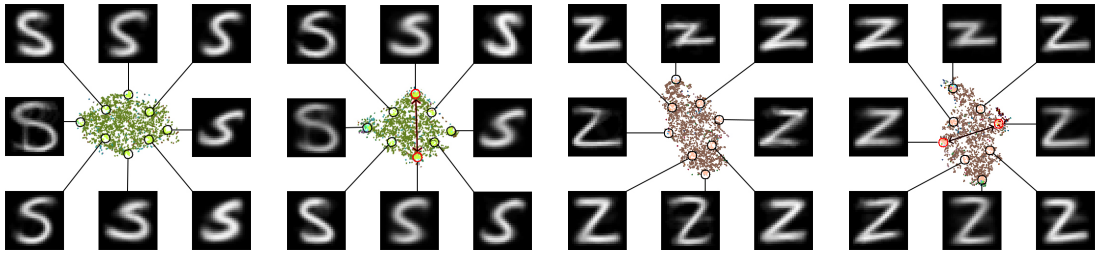


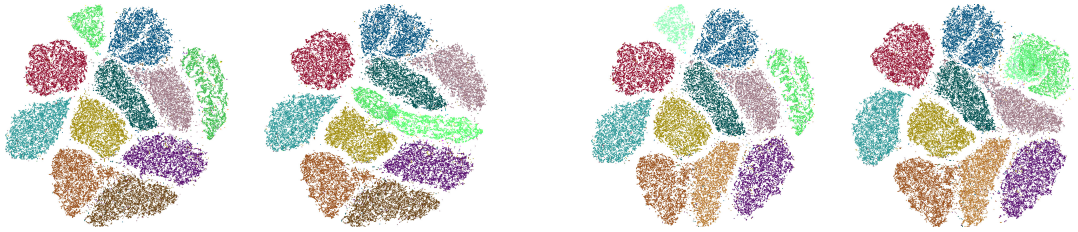
Figure 5: Shaping the cluster of the class of `1`s in the MNIST dataset by dragging 1's written slanted to the left and 1's written slanted to the right to different locations and letting the cluster adapt. (top) The resulting cluster within the larger embedding when putting the two endpoints on a horizontal line. (bottom) The left side shows all datapoints of class `1` isolated, prior to fixing any datapoints. The right side shows the two endpoints of obliqueness dragfixed in opposite corners of the embedding space, with the rest of the cluster adapting.

and then the cluster will – with some tweaking of weight and falloff settings – adapt to these two extremes. We show the cluster in Figure 5. This only works for properties with respect to which the kNN graph is structured and, as a result, the graph already shaped. That is, we simply grab points along the cluster's edge and dragfix those to specific locations. Figure 6 shows an example where dragfixing manages two flip two clusters by swapping two points on their edge. However, shaping a cluster with respect to some non-primary characterstic – on which the kNN graph is not based – is more difficult, if not impossible. For example, line thickness



(a) ***s*** (b) ***s***, top/down swapped (c) ***z*** (d) ***z***, left/right swapped

Figure 6: Two clusters, both showing their initial form and their form when two opposing points on the edge are swapped. Both clusters adapt as expected.



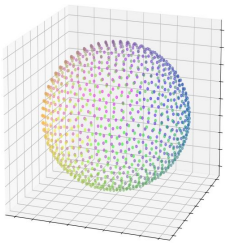
(a) Double-selection (b) Fusion

Figure 7: Two methods for merging clusters. (a) Each cluster is selected with a separate selection (indicated by the slightly different shades of green) and only the similarities between neighbours that are from different clusters are multiplied. (c) The clusters are merged by fusion (i.e., adding links between datapoints from the two selections). This offers little advantage in this scenario, since there already exist sufficient links between the two clusters.

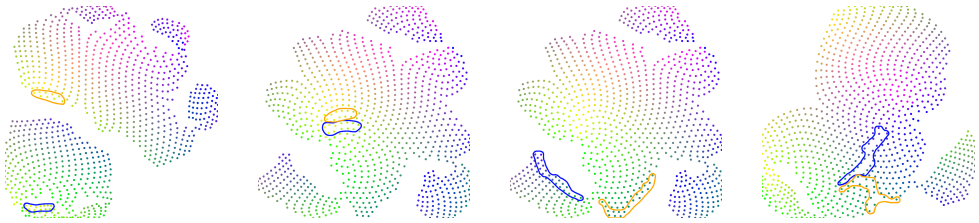
is a human-identifiable property that does not smoothly vary across the cluster, but varies much more haphazardly, with different regions of the cluster exhibiting similar line thickness. The extremes of this property are not located on the edges of the cluster, but somewhere within.

4.2 Cluster merging

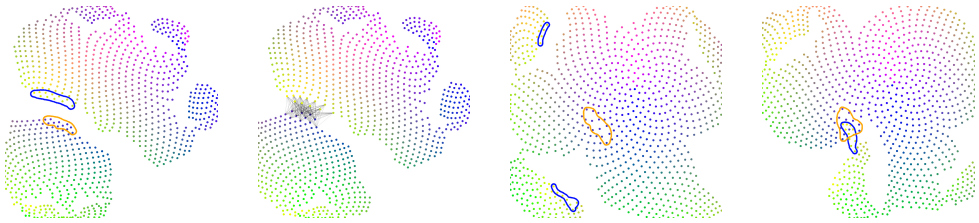
Figure 7 demonstrates cluster merging on an erroneously separated cluster in MNIST with both double-selection cluster merging and fusion. Both methods are effective, because there already exist sufficient links between the two subclusters. For demonstrating the effectiveness of fusion, we use points on a three-dimensional sphere, which we briefly mentioned as example in the introduction to Section 3. Embedding such a sphere on a two-dimensional plane necessarily involves cuts across the sphere's surface. Because the three-dimensional points are evenly distributed across the sphere, it is completely arbitrary where these cuts are made. Our techniques offers control over where the cuts should not be made. Figure 8 presents two examples. In the first, the points in the different selections lie next to each other on the sphere's surface, and fusing the selections sews an actual cut. The two selections are already relatively densely interconnected, and the fusing operation adds only a handful of new links, but those are suffi-



(a) Points on a sphere in 3D



(b) Two steps of fusing "natural" cuts in the sphere's embedding, with the two selections encompassing points that are adjacent on the sphere's surface.



(c) Fusing two selections that encompass separated regions of the sphere's surface.

Figure 8: Merging by fusion demonstrated on two different kind of cuts made between evenly distributed points on a three-dimensional sphere.

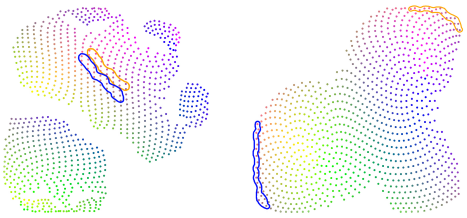


Figure 9: Separating datapoints in an embedding of a dataset of points on a sphere. By setting similarities to zero, the user can designate where cuts should be placed.

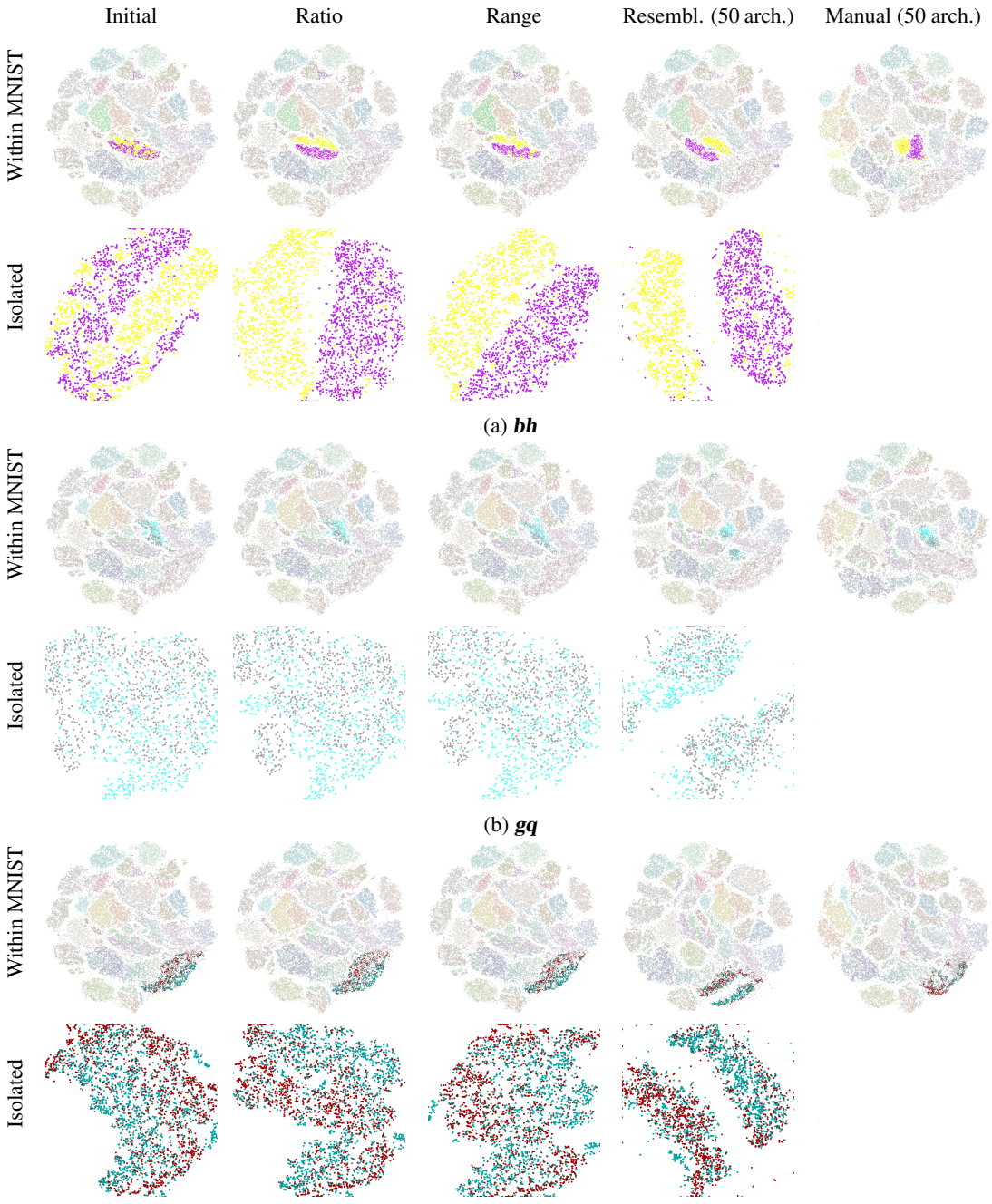
cient to make sure that the cut is sewn, at the cost of another cut made elsewhere. The resulting seams are correctly stitched together and seem natural within the embedding. In the second example, we choose to stitch together arbitrary parts of the edge; parts that aren't co-located on the sphere's surface and between which are no interselection links. Fusing creates these links, but they are not strong enough to unite the selections. We therefore double the similarities along those links. This results in merged selections, although the stitching looks awkward compared to the natural-looking stitching of the first example. Even though this is a toy example, the problem of t-SNE needing to induce distortions in some form in order to collapse dimensions applies to general, higher dimensional datasets as well.

4.3 Cluster separation

The most effective way to separate groups of datapoints is to set the intergroup similarities to zero or to remove the interlinks entirely, but this is viable only when the similarities to be severed lie along a clearly distinguishable dividing seam, and the two groups of datapoints that define that seam can be easily selected. Figure 9 returns to the sphere dataset from the previous section. Besides designating where cuts should not be placed, we can designate where cuts should be placed by selecting both sides of the cut and setting interselection similarities to zero.

For clusters where datapoints from different classes have fully mixed, the above method cannot be used. We found three such clusters in our embedding of the EMNIST dataset [9], namely a cluster of lowercase ***b***'s and ***h***'s, a cluster of lowercase ***g***'s and ***q***'s, and a cluster of lowercase ***i***'s and ***l***'s. Figure 10 shows the three methods of cluster separation and the manual method on the three clusters. Each method of separation is shown executed on each cluster both with the rest of the dataset – i.e., with influence from other EMNIST datapoints – and in an isolated setting, with only the relevant datapoints and forces present.

Ratio, range, and – for the most part – resemblance-based cluster separation are designed to focus on a user-indicated subset of discriminatory attributes when adapting similarities. For the methods to be effective, they require such a subset to exist and to be clearly distinguishable and easily selectable. Image datasets are an obvious example of datasets with such attributes, because of their straightforward visualization, in which texels can be easily selected. However, images can be blurry, noisy, and/or vary in composition to such an extent that there is little overlap between images in terms of the texels that display the discriminatory features. In that case, the images do not exhibit a consistent subset of discriminatory attributes, rendering the methods ineffective.
Unfortunately, this is largely the case for EMNIST. The ***bh*** cluster is the only cluster we have found on which the three separation methods are effective. Figure 11 illustrates why; it is the only cluster on which our selection and the data agree on the set of discriminatory attributes. For the other two clusters, the data is too noisy, causing the average over all differences to be similar between interclass and intraclass neighbours. In other words, using only the differences does not provide enough information for effective separation of the ***gq*** and ***il*** clusters. Appendix A gives a numerical evaluation of the three methods. The manual method is superior, though it requires significant user input. The resemblance and manual separations in Figure 11 were obtained with 25 archetypes per class for a total of 50 archetypes, and minimal user input during the separation

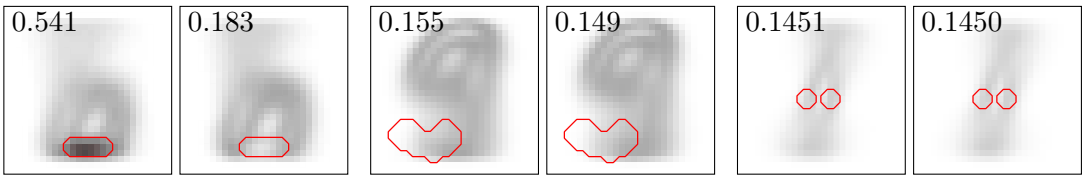


(a) ***bh***

(b) ***gq***

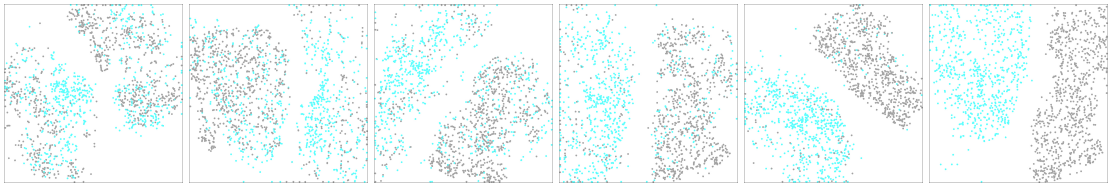
(c) ***il***

Figure 10: The four methods of separating executed on three different clusters that each consist of two classes, captured after 1,000 iterations. There are no results for the manual mode of separation in isolated settings, because the implementation we use is not suited for such a small number of datapoints.



(a) ***bh*** (b) ***gq*** (c) ***il***

Figure 11: The per-attribute difference averaged between all selected interclass neighbours (left) and intraclass neighbours (right) for each of the three clusters. Our selection of attributes are indicated with a red outline. The average of these outlined differences is in the top left corner of the corresponding picture. We note that the number of ***i***'s written with a separate line and dot is very small in EMNIST, and the main difference between ***i***'s and ***l***'s seems to be the thickness of the (singular) line, hence our attribute selection.



(a) 50 arch. (b) 160 arch. (c) 420 arch. (d) 841 arch. (e) 1,263 arch. (f) 1,680 arch.

Figure 12: Using the resemblance-based method of separation on the isolated ***gq*** cluster with an incrementing number of archetypes. The quality of the separation increases rather linearly with the number of archetypes. (f) With one archetype per datapoint, the method naturally manages a perfect separation.

mode. Figure 12 shows how the quality of separation increases more or less linearly with the number of archetypes. Ideally, little user effort would already result in qualitative separations, but that is not the case. We emphasize that our method is interactive, such that separation can be iteratively refined by repeated splitting and merging.

On the other hand, even datasets for which more simple visualization idioms (e.g. barplots/histograms, radar charts, parallel coordinates plots, etc.) are the most natural visualization can exhibit distinct and easily selectable discriminatory attributes. For example, a hyperspectral image can be regarded as a dataset, with the texels in the image as datapoints and the wavelength bands that partition the electromagnetic spectrum as attributes. Texels that represent different metals (iron, copper, lead, etc.), or different kinds of organic surface (grass, soil, foliage, etc.), could be erroneously clustered together. If the attributes are visualized with a histogram ordered by wavelength, our separation methods could be applied by selecting the discriminatory wavelength bands in the histogram with relative ease.

4.4 Case studies

In this section, we focus on two other datasets: COIL-20 in Section 4.4.1 and Fashion MNIST in Section 4.4.2. We demonstrate how their embeddings can be made more intuitive using a

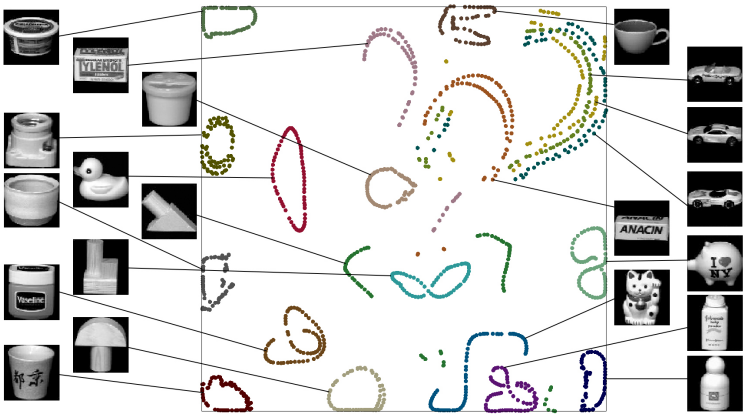
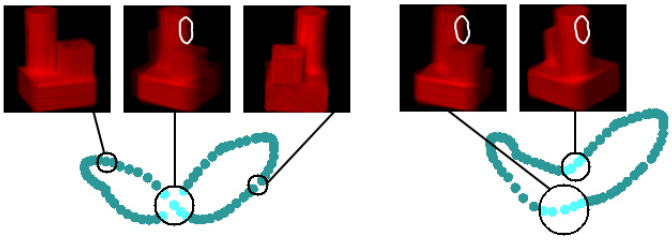


Figure 13: An embedding of the COIL-20 dataset [23] generated by our implementation of t-SNE with a perplexity of 30 and $k = 91$.

combination of the above techniques.

4.4.1 COIL-20

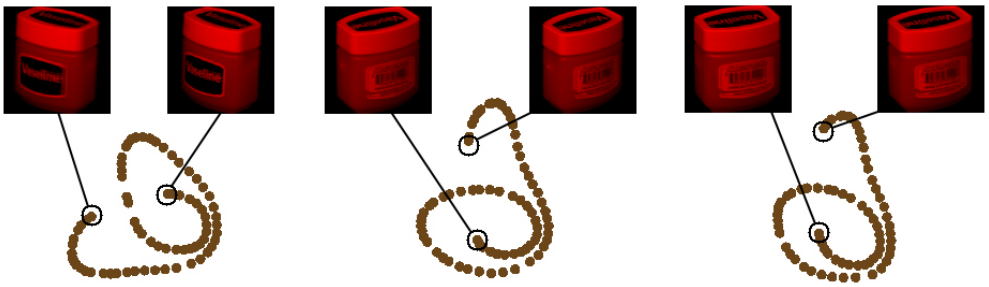
The Columbia University Image Library 20 (COIL-20) dataset contains 1,440 128×128 grayscale images of tabletop objects [23]. There are 20 object classes. Each object was placed on a motorized turntable against a black background and photographed every 5 degrees during a 360 degree spin, resulting in 72 images per object (class). The embedding is pictured in Figure 13. The most intuitive embedding would consist exclusively of a closed, simple, non-intersecting loop of datapoints per class, with all the loops evenly distributed across the two-dimensional space. t-SNE succeeds at this for many classes. We discuss some of the less successful ones.



(a) (b)

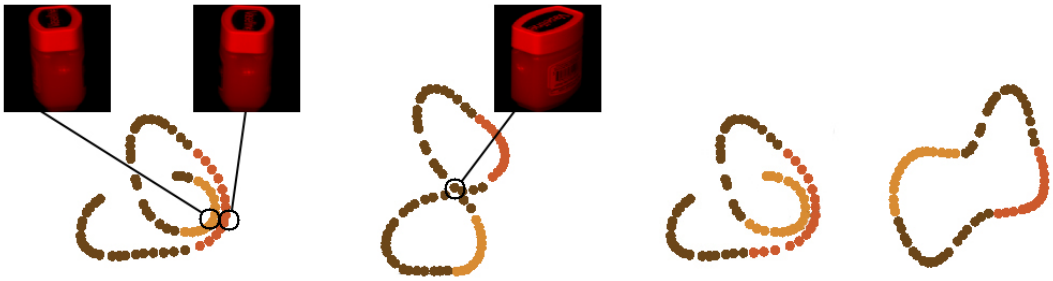
The above loop corresponds to a wooden factory-like toy block. The loop in the embedding without user manipulation is pictured in (a) and intersects itself, resembling an ∞-shape. This is unsurprising on closer inspection of the intersection, where the object resembles its 180° rotated equivalent. Moreover, this is apparently the orientation that most resembles its 180° rotated equivalent out of all possible orientations. This makes sense, because the cube and cylinder on top of the base are more or less lined up. Luckily, the spacing between the datapoints at the intersection indicates that the opposites are still not too similar, and the discriminatory attributes can still be distinguished. We chose the attributes delineated within the white outline, and based on those, we separated the selected (and in the pictures highlighted) datapoints using the range-

based method. This results in the simple loop shown in (b), where the datapoints that took part in the operation are still selected. The ratio and resemblance-based method are both also effective.



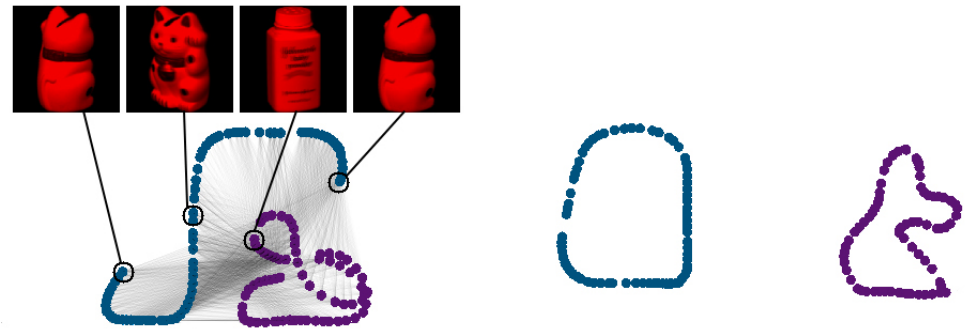
(a) (b) (c)

Next, we look at a loop representing a package of brand petroleum jelly. Starting out, there is a gap in the loop through which it crosses itself, as shown in (a). It is very difficult to close this gap by selecting the two endpoints and slowly increasing their similarity until they meet, since a new gap will open somewhere else. Two consecutive attempts are shown in (b) and (c).



(a) (b) (c) (d)

The problem is the two segments at the bottom right that stick together. Below, we inspect these datapoints and see that this is another case where the objects in these orientations are highly similar to their $180°$ rotations. The two segments are easy to select with two distinct selections, shown by the two different hues of brown above. Separating the selections by severing the links results in the ∞-shaped loop next to it. The intersection is simply an extension of the two neighbouring segments in the original loop that we neglected to separate, so we can either use the range-based method that we used on the toy factory loop above, or we select more of the adjacent segments and separate those, which results in a simple loop.



(a) (b) (c)

Some loops are distorted due to attractive forces from other loops. The above line represents a Maneki-neko, a common Japanese figurine of a cat believed to bring good luck. It is broken up at a rather arbitrary location along the loop; rerunning the minimization often breaks the loop at other points. The reason for this is its tendency to wrap around the purple loop, consisting of images of a bottle of baby powder. This bottle has roughly the same dimensions and shape, and at some orientations the outlines of the two objects are highly similar, so their gravitational pull is unsurprising. Selecting the two loops with different selections and severing the interlinks results in the simple loop for the Maneki-neko figurine in (b).
The loop of baby powder is more difficult to untangle. The reason for this are the multiple orientations that resemble other orientations due to the two lines of symmetry. It requires a combination of the previously discussed techniques in multiple steps and careful selection of datapoints, but it is feasible, with our result shown in (c).

(a) (b) (c)

Likewise, the green loop selected above, corresponding to a triangular wooden toy block, is fractured because small chunks of it are attracted to the two loops discussed above. Again, the objects have very similar sizes and outlines at the orientations of the images of which the datapoints lie close to each other. We could unite the datapoints into one simple loop by severing the interloop links again – although this does require an additional step of increasing the loop's intrasimilarities – but we want to demonstrate an alternative method. The problem boils down to the fact that, in the high dimensional space, some datapoints in the loop have picked undesirable nearest neighbours, namely those from other classes. We can fix this by first severing all intralinks between the selected datapoints, leaving the embedding in the state pictured in the middle. We then fuse every pair of selected datapoints and sever all links between the selected

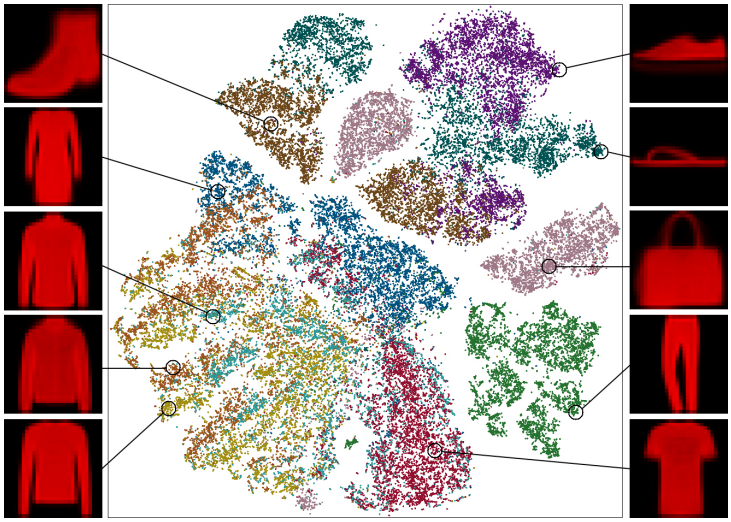
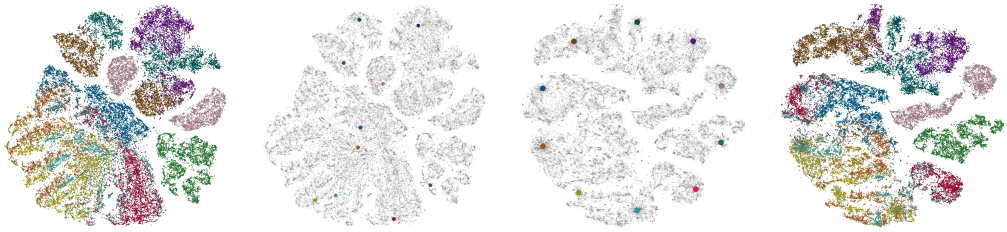


Figure 14: An embedding of the Fashion MNIST dataset [37] generated by our implementation of t-SNE with a perplexity of 30 and $k = 91$.



(a) (b) (c) (d)

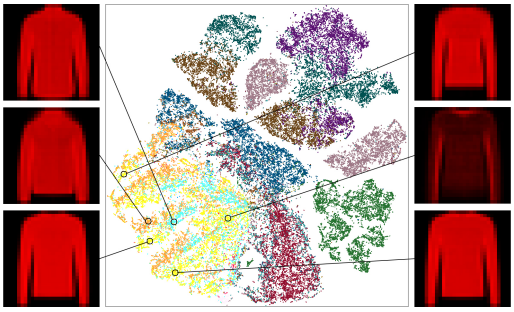
Figure 15: (a) t-SNE embedding of Fashion-MNIST. (b) Labeled-only selection mode is enabled. (c) The labeled datapoints are fixed and arranged in a circle, similarly to Figure 4(a). (d) The embedding tries to adapt to the fixed datapoints, but the embedding does not improve in terms of readability.

datapoints and all other datapoints in the embedding, which yields a simple loop as shown in the top right of the right picture.

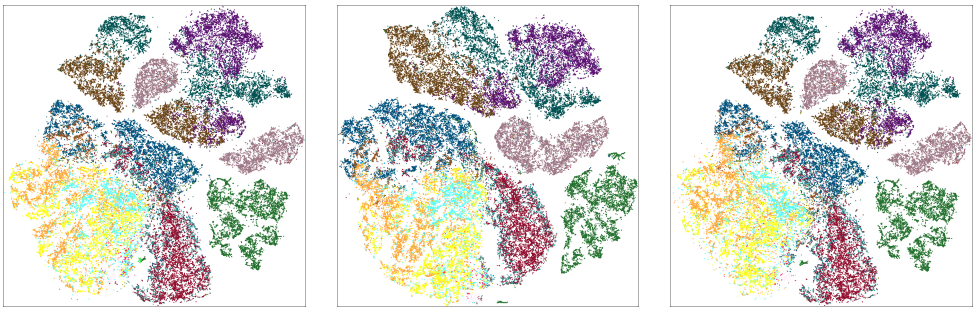
4.4.2 Fashion MNIST

Fashion MNIST is a dataset published by Zalando research and similar to MNIST in size and dimensionality, namely 60,000 grayscale 28×28 images, but depicting fashion items instead of handwritten digits [37]. Each item belongs to one of ten categories, which is the same number of classes as MNIST. It was compiled in 2017 as a more challenging variant of MNIST. This is evident when projecting the dataset with t-SNE; Figure 14 shows an embedding of Fashion MNIST with our implementation. The clusters are not as clearly defined as the embeddings of regular MNIST. Figure 15 shows dragfixing of one datapoint per class to a specific location in the hopes that the rest of the datapoints follow suit, which was the case for regular MNIST as

demonstrated in Figure 4, but which does not happen for Fashion MNIST.

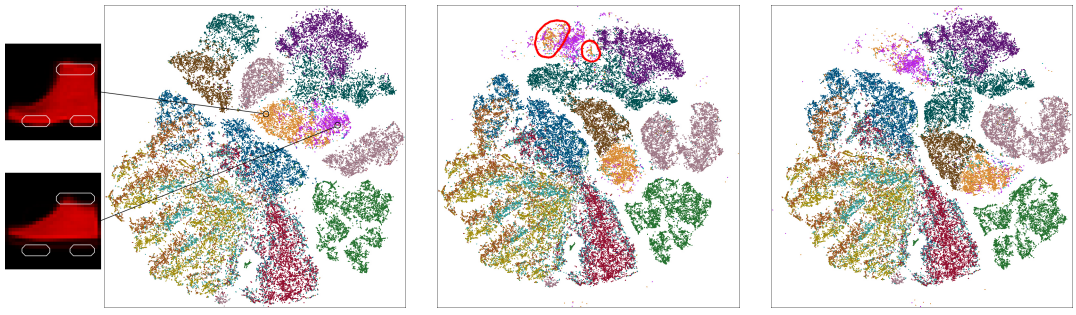


(a)



(b) (c) (d)

In (a) above, the large cluster at the bottom left is selected. It consists of datapoints from the categories `Pullover`, `Coat` and `Shirt`; all upper body wear with long sleeves. The datapoints are not fully mixed, but rather multiple uniclass subclusters have formed and those have mixed. This clustering seems to be based on differences in intensity values and slight differences in outlines, making it practically impossible to identify discriminatory attributes. With the aim of a three-way separation, we resort to the manual mode of separation. (b) shows the result with 10 archetypes per class, (b) shows the result with 25 archetypes per class, and (c) shows the result with 50 archetypes per class. 10 archetypes per class appear too few to make a marked difference compared to the original embedding. With 25 archetypes per class, a clearer separation into three main clusters appears to manifest. The 50-archetype result is slightly better still, but arguably does not warrant the increase in user effort.



(a) (b) (c)

The cluster selected in (a) consists of tall Sneakers and short Ankle boots. Both are approximately halves of their respective class, with short Sneakers and tall Ankle boots having landed elsewhere in the embedding (namely in the top right and top left). Datapoints from the different classes have not fully blended and are relatively well separated, and a careful double selection and severing of interlinks is effective, if a bit time-consuming. For purposes of demonstration, we again use resemblance-based separation. The ratio and range-based methods of separation are ineffective in this scenario. We can identify discriminatory attributes, namely those that are most dependent on the size of the shoe. We highlighted our choice of attributes in the attribute visualization. (b) shows the result after applying the method once, with 25 archetypes per class. It is a fairly good separation, and the (selected) cluster of short Ankle boots unites with its tall Ankle boot counterpart. However, there still are two sizable segments of Ankle boots among the Sneakers. We select these segments, highlighted with a red outline, add ten more archetype from suggestions based on this selection (for a total of 35 archetypes per class), and retry the procedure. This results in the second separation shown in (c), where the presence of Ankle boots among the Sneakers is markedly reduced.

5 Discussion

Projections of high-dimensional datasets produced by t-SNE cannot – in practical scenarios – reflect high-dimensional neighbourhoods, causing unavoidable projection errors. Our work focused on allowing data analysts to make embeddings more representative with user-imposed constraints on projected datapoint locations and manipulations of the probability distribution of the high-dimensional distances. To this end, we proposed a variety of techniques, each tailored to mitigate different ways in which embeddings may incorrectly reflect the high-dimensional data and hinder their intuitiveness. We explained our proposals in Section 3 and highlighted situations in which their application is effective in Sections 4. This section offers a discussion of the results and avenues for future research.

All methods alter the manifestation of projection errors of embeddings, but the extent to which this increases their accuracy of representation depends on which method is applied to what data and/or problem, as our experiments anecdotally demonstrate. The most simple techniques are applicable on any type of dataset and are always effective: selecting, disabling and isolating datapoints, visualizing links, cluster merging, and cluster separation by severing links. Dragfixing datapoints and manual cluster separation are still applicable to any type of dataset, but their effectiveness is dependent on the data. The usefulness of dragfixing singular datapoints depends on how well t-SNE already clusters the data. For instance, MNIST's embedding is well-clustered and dragfixing one datapoint per class is effective for approximate repositioning of the corresponding clusters, but Fashion MNIST is less well-separated, making dragfixing singular datapoints not nearly as useful. Clusters can be shaped with simple transformation by dragging points on the edge into desired configurations, but it is difficult to shape them according to some non-primary characterstic to which the cluster is not already shaped.

During the fusion operation for merging clusters, the dropout of candidate neighbours is done deterministically based on a modulo operation on datapoint indices. A better way is likely to, for any given datapoint in one selection, enumerate and sort the Euclidean distances to all datapoints in the other selection, add only the closest k datapoints as neighbours (if they aren't already), and symmetrize both their kNN sets. However, we opted for a less complex implementation.

The three methods of cluster separation are only relevant and effective for embeddings that:
1) have incorrectly mixed datapoints that belong to different classes or are otherwise distinct.
2) those datapoints exhibit a generally consistent subset of discriminatory attributes on which they differ to a sufficiently significant degree.
3) those discriminatory attributes are easy to distinguish and select.
These conditions limit the applicability of the separation techniques. We exclusively used grayscale image datasets, which are natural candidates, but even among those, the applicability depends on the type of images. For example, MNIST and EMNIST have clearly defined outlines and depict recognizable symbols, but Fashion MNIST depicts real-life objects of which humans have an idea of the general shape, but which often divert from this idea. This makes it difficult to distinguish discriminatory attributes, which renders those techniques ineffective. We found only two instances where our separation methods proved effective: a cluster of lowercase ***b***'s and ***h***'s in EMNIST, and a cluster/loop of a toy block in COIL-20.

5.1 Future research

The main attraction of this work is the set of cluster separation methods. While we exclusively used image datasets for showcasing these methods, tailor-made, intuitive visualizations can be developed for other fields of research as well, as long as these visualizations support the identification and selection of discriminatory attributes. Different subsets of attributes could be visualized with different idioms. A natural question to ask is whether our proposed techniques can be extended to suit any type of dataset, with arbitrary attributes and regardless of their visualization, for example through automatic detection of the discriminatory attributes.

Our focus was on the implementation of functionality supporting our objective statement. However, this functionality can be extended with features that further streamline the user experience and/or help the user gain more insight from embeddings. To name a few suggestions: zooming and panning, on-the-fly labeling of datapoints, more advanced filtering on and searching of datapoints based on attribute values, better ways of visualizing the magnitude of similarities corresponding to links in the kNN graph, and interactive manipulation of these kNN graph links, e.g., with a knife tool that cuts links, a click-and-drag method to create new links, or a select-and-scroll method to adjust similarities.

Acronyms

BH t-SNE Barnes-Hut t-SNE. 8

A t-SNE Approximated t-SNE. 8

BH Barnes-Hut. 8, 32

COIL-20 Columbia University Image Library 20. 23, 24, 30

DH t-SNE Dual-Hierarchy t-SNE. 8

EMNIST Extended Modified National Institute of Standards and Technology. 17, 21, 23, 30

GPGPU general purpose computing on graphics processing units. 16

GPU graphics processing unit. 8

HSNE hierarchical stochastic neighbourhood embedding. 8

KL Kullback-Leibler. 6

kNN k nearest neighbour. 8, 11, 16–18, 30, 31, 41

L t-SNE Linear t-SNE. 8

LLE Locally Linear Embedding. 5

MDS multidimensional scaling. 5

MNIST Modified National Institute of Standards and Technology. 17–19, 23, 27, 28, 30

PCA principal component analysis. 4, 5

SNE Stochastic Neighbor Embedding. 5, 6

t-SNE t-distributed Stochastic Neighbor Embedding. 4–6, 8–13, 16–18, 21, 24, 27, 30, 32, 33

UMAP Uniform Manifold Approximation and Projection. 5

Glossary

attribute See dimension. 4, 9, 10, 12–15, 21, 23, 24, 30, 31, 33

class One of a number of distinct conceptual categories that can be assigned to datapoints. 10–12, 14, 15, 17, 21, 22, 24, 26–30, 38, 40, 42

data visualization The field of computer science that aims to enhance the effectiveness of visual representations of datasets. Sometimes referred to as visual analytics. 4

datapoint The same as a record, but this term is used more in a Euclidean spatial context, where records are locations in some space. 4–31, 33, 38–40, 42

dataset A collection of data, often interpreted as a table, where each row is a records, also called datapoints, samples, or entries, and each column is a dimension, also called a attribute or feature. 4, 5, 9, 10, 12, 16, 17, 21, 23, 24, 27, 30, 31, 33

dimension A (single-value) descriptor or characterstic of a data entry/sample. Also called an attribute. 4, 9, 21, 33, 34

dimensionality reduction The act of reducing the number of dimensions of a (high-dimensional) dataset. 4, 5, 8, 9, 33

early compression An optimization technique for t-SNE that adds an L2-penalty that penalizes long distances of the datapoints to the origin, forcing the datapoints to stay close together and move through each other. 7

early exaggeration The collection of dimensionality reduced datapoints, or, the mapping of the high-dimensional dataset to the low dimension. 7

embedding The collection of dimensionality reduced datapoints, or, the mapping of the high-dimensional dataset to the low dimension. Also referred to as projection. 4–6, 8–11, 16–18, 20, 21, 23, 24, 26–31, 34

entry See record. 33

feature See dimension. 33

ground truth The factually correct result of any machine learning problem instance, or in the context of non-linear dimensionality reduction, the correct classification of datapoints provided by direct observation and measurement rather than inference. In the context of machine learning, this refers the the manually annotated labels of datapoints, denoting the actual class is belongs to. Machine learning algorithms that train with such an annotated dataset are called supervised learning algorithms; if such labels are not provided, this is called unsupervised learning. 38, 39, 42

high-dimensional The property of data that indicates that the number of dimensions is larger than three. Some definitions formulate it as the property of data that indicates that the number of dimensions exceeds the sample size (the number of data entries). 4, 7, 30

low-dimensional The property of data that indicates that the number of dimensions remains tractable for a given purpose, be it visualization or otherwise. 4

manifold A topological space that locally resembles Euclidean space near each point. 5

objective function A function which an algorithm should either maximize (in which case it is also called a goal function) or minimize (in which case it is also called a cost function) in gradient descent algorithms. 6, 7

pairwise Considering every possible pair of two elements within a set of elements. 5

perplexity Roughly corresponds to the continuous analogue to k in k nearest neighbour techniques. 6

projection Same as embedding. 4, 5, 33

record An item in a dataset. Alternatively, a point in the space spanned by the dataset's dimensions. 33, 34

sample See record. 33

References

[1] S. J. Aarseth and S. J. Aarseth. *Gravitational N-body simulations: tools and algorithms*. Cambridge University Press, 2003.

[2] H. Abdi and L. J. Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[3] E.-a. D. Amir, K. L. Davis, M. D. Tadmor, E. F. Simonds, J. H. Levine, S. C. Bendall, D. K. Shenfeld, S. Krishnaswamy, G. P. Nolan, and D. Pe'er. visne enables visualization of high dimensional single-cell data and reveals phenotypic heterogeneity of leukemia. *Nature biotechnology*, 31(6):545–552, 2013.

[4] J. Barnes and P. Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986.

[5] B. Becher, A. Schlitzer, J. Chen, F. Mair, H. R. Sumatoh, K. W. W. Teng, D. Low, C. Ruedl, P. Riccardi-Castagnoli, M. Poidinger, et al. High-dimensional analysis of the murine myeloid cell system. *Nature immunology*, 15(12):1181–1189, 2014.

[6] I. Borg and P. J. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[7] D. M. Chan, R. Rao, F. Huang, and J. F. Canny. t-sne-cuda: Gpu-accelerated t-sne and its applications to modern data. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 330–338. IEEE, 2018.

[8] A. Chatzimparmpas, R. M. Martins, and A. Kerren. t-viSNE: Interactive assessment and interpretation of t-SNE projections. *IEEE transactions on visualization and computer graphics*, 26(8):2696–2714, 2020.

[9] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. Emnist: an extension of mnist to handwritten letters, 2017.

[10] G. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *NIPS*, volume 15, pages 833–840. Citeseer, 2002.

[11] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2021.

[12] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. Chau. A cti v is: Visual exploration of industry-scale deep neural network models. *IEEE transactions on visualization and computer graphics*, 24(1):88–97, 2017.

[13] S. Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[14] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[15] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[16] N. Li, V. van Unen, T. Höllt, A. Thompson, J. van Bergen, N. Pezzotti, E. Eisemann, A. Vilanova, S. M. Chuva de Sousa Lopes, B. P. Lelieveldt, et al. Mass cytometry reveals innate lymphoid cell differentiation pathways in the human fetal intestine. *Journal of Experimental Medicine*, 215(5):1383–1396, 2018.

[17] A. Mahfouz, M. van de Giessen, L. van der Maaten, S. Huisman, M. Reinders, M. J. Hawrylycz, and B. P. Lelieveldt. Visualizing the spatial gene expression organization in the brain through non-linear similarity embeddings. *Methods*, 73:79–89, 2015.

[18] S. Marsland. *Machine learning: an algorithmic perspective*, chapter 6. Chapman and Hall/CRC, 2 edition, 2014.

[19] R. M. Martins. *Explanatory visualization of multidimensional projections*. PhD thesis, Universidade de São Paulo, 2016.

[20] L. McInnes, J. Healy, and J. Melville. Umap: uniform manifold approximation and projection for dimension reduction. 2020.

[21] A. Mead. Review of the development of multidimensional scaling methods. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 41(1):27–39, 1992.

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[23] S. A. Nene, S. K. Nayar, H. Murase, et al. Columbia object image library (coil-20). 1996.

[24] H. Perez and J. H. Tah. Improving the accuracy of convolutional neural networks by identifying and removing outlier images in datasets using t-sne. *Mathematics*, 8(5):662, 2020.

[25] N. Pezzotti, B. P. Lelieveldt, L. Van Der Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and user steerable tsne for progressive visual analytics. *IEEE transactions on visualization and computer graphics*, 23(7):1739–1752, 2016.

[26] N. Pezzotti, J. Thijssen, A. Mordvintsev, T. Höllt, B. v. Lew, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. GPGPU Linear Complexity t-SNE Optimization. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of VAST 2019)*, 26(1):1172–1181, 2020.

[27] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[28] K. Shekhar, P. Brodin, M. M. Davis, and A. K. Chakraborty. Automatic classification of cellular expression by nonlinear stochastic embedding (accense). *Proceedings of the National Academy of Sciences*, 111(1):202–207, 2014.

[29] M. Van de Ruit, M. Billeter, and E. Eisemann. An efficient dual-hierarchy t-sne minimization. *IEEE Transactions on Visualization and Computer Graphics*, 2021.

[30] L. Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

[31] L. Van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

[32] V. van Unen, T. Höllt, N. Pezzotti, N. Li, M. J. Reinders, E. Eisemann, F. Koning, A. Vilanova, and B. P. Lelieveldt. Visual analysis of mass cytometry data by hierarchical stochastic neighbour embedding reveals rare cell types. *Nature communications*, 8(1):1740, 2017.

[33] V. M. Vu, A. Bibal, and B. Frénay. Integrating constraints into dimensionality reduction for visualization: A survey. *IEEE Transactions on Artificial Intelligence*, 3(6):944–962, 2022.

[34] M. Wattenberg, F. Viégas, and I. Johnson. How to use t-sne effectively. *Distill*, 1(10):e2, 2016.

[35] Wikipedia contributors. Kullback–Leibler divergence — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Kullback%E2%80%93Leibler_divergence&oldid=1043560273`, 2021. [Online; accessed 11-September-2021].

[36] J. Xia, Y. Zhang, J. Song, Y. Chen, Y. Wang, and S. Liu. Revisiting dimensionality reduction techniques for visual cluster analysis: An empirical study. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):529–539, 2021.

[37] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Appendix A Numerical evaluation of automatic separation methods

Figure 16 and Table 1 show why resemblance-based cluster separation works better than the other two similarly performing methods, though not by much. We employ three metrics to quantify the quality of the resemblance-based separations:

- *Purity*: the percentage of datapoints whose class is the majority class of the cluster they are in. It measures the extent to which clusters contain a single class. It is calculated by identifying the majority class per cluster, summing the datapoints belonging to the majority class over all clusters, and dividing this by the total number of datapoints:

$$\text{Purity} = \frac{1}{n'} \sum_{c \in C} \max_{l \in L} |c \cap l| \tag{17}$$

  where C is the set of relevant clusters, L is the set of classes (the 'l' is for 'label'), n' is the number of datapoints in all of C's clusters, and $\max_{l \in L} |c \cap l|$ is the number of datapoints of the majority class of cluster c. In our case of two classes, purity ranges from 0.5 to 1.

  The purity measure has two drawbacks for general cluster evaluation. First, the measure does not penalize having many clusters[1]. However, the resemblance-based separation method consistently separates the three clusters we investigate into two clusters. Secondly, the metric does not work well for imbalanced class shares; the larger the class imbalance, the larger the lower bound on the metric range. Though the three clusters are imbalanced, their degree of imbalance is low and roughly the same (with ratios ranging between 1.15 to 1.18).

- *Adjusted rand index* (ARI): uses the notion of true and false positives and negatives – from the fields of machine classification and detection – to measure the extent to which datapoints are correctly clustered together. The metric considers pairs of datapoints and considers a pair classified as "positive" if they are clustered together and "negative" if they are in separate clusters. The ground truth labels can tell us whether these classification are correct, or "true", or incorrect, "false". The classification results are usually collected in a 2×2 *confusion matrix*, with the top row containing true positives and false negatives, and the bottom row containing false positives and true negatives. The original rand index is calculated as follows:

$$\text{RI} = \frac{TP + TN}{TP + FP + TN + FN} \tag{18}$$

  which is simply the ratio of correct classifications to all classifications. The rand index is sensitive to chance; it does not take into account the fact that a completely random

[1] A purity of 1 is always possible by having each datapoints in its own singleton "cluster".

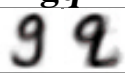
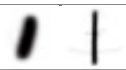
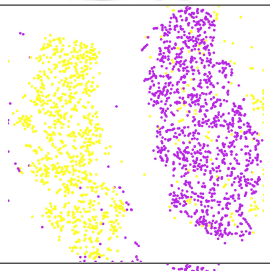
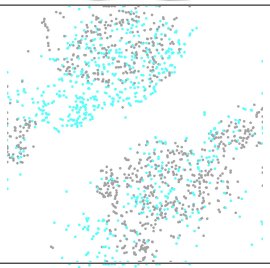
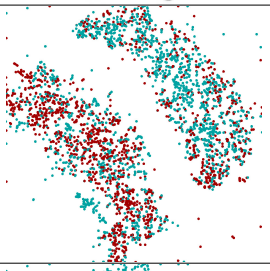
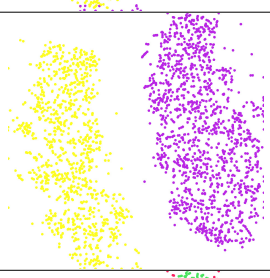
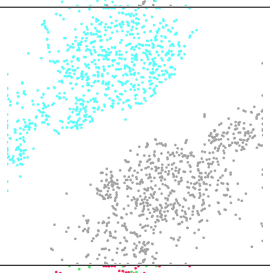
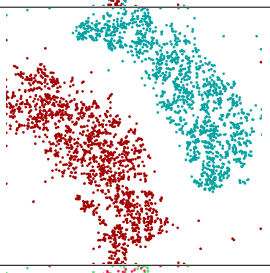
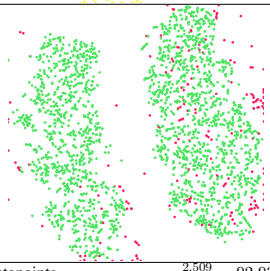
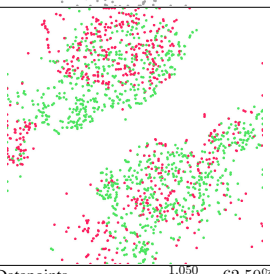
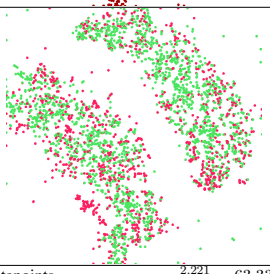
Cluster	***bh***	***gq***	***il***
Archetypes			
Actual labels			
Classified labels			
Correctness			
Correctness in %'s	Datapoints $\frac{2,509}{2,700} = 92.93\%$ Neighbours $\frac{301,324}{338,108} = 89.12\%$ - Neighbours (inter) $\frac{64,060}{81,332} = 78.76\%$ - Neighbours (intra) $\frac{237,264}{256,776} = 92.40\%$	Datapoints $\frac{1,050}{1,680} = 62.50\%$ Neighbours $\frac{122,760}{218,048} = 56.30\%$ - Neighbours (inter) $\frac{26,322}{87,232} = 30.17\%$ - Neighbours (intra) $\frac{96,438}{130,816} = 73.72\%$	Datapoints $\frac{2,221}{3,507} = 63.33\%$ Neighbours $\frac{248,420}{435,664} = 57.02\%$ - Neighbours (inter) $\frac{34,518}{182,016} = 18.96\%$ - Neighbours (intra) $\frac{213,902}{253,648} = 84.33\%$
Contingency table	Clusters \ Classes: ***b*** / ***h*** / Total Top: 141 / 1409 / 1,550 Bottom: 1,100 / 50 / 1,150 Total: 1,241 / 1,459 / 2,700	Clusters \ Classes: ***g*** / ***q*** / Total Top-left: 533 / 333 / 866 Bottom-right: 237 / 577 / 814 Total: 770 / 910 / 1,680	Clusters \ Classes: ***i*** / ***l*** / Total Left: 741 / 1,077 / 1,818 Right: 1,137 / 552 / 1,689 Total: 1,878 / 1,629 / 3,507
Clustering evaluation	Purity 92.93% ARI 0.737 AMI 0.639	Purity 66.07% ARI 0.102 AMI 0.077	Purity 63.13% ARI 0.069 AMI 0.052

Table 1: A comparison of the resemblance-based separation results of the three clusters. The first row shows the archetypes used on each cluster by the method. The second row shows the ground truth labeling of each cluster. The third row shows the classification of the datapoints. The fourth row shows the correctness of the method's classification, with green datapoints indicating a correct classification (a match between classification and label) and red indicating an incorrect classification (a mismatch between classification and label). The fifth row contains some statistics regarding the correctness of the classification: the percentage of correctly classified datapoints, the percentage of correctly classified neighbour relationship types (interclass or intraclass) based on the datapoint label classifications, the percentage of interclass neighbour pairs that were correctly classified as interclass, and the percentage of intraclass neighbours that were correctly classified as intraclass. The sixth row contains contingency matrices as subtables. The seventh and last row contains the (external) clustering evaluation metrics that are based on the contingency tables.

clustering may still correctly cluster datapoints by accident. The adjusted rand index solves this. It is calculated as:

$$\text{ARI} = \frac{RI - \text{E}(RI)}{\max(RI) - \text{E}(RI)} \tag{19}$$

Using the confusion matrix directly, it can be calculated as:

$$\text{ARI} = 2\frac{TP * TN - FN * FP}{(TP + FN) * (FN + TN) + (TP + FP) * (FP + TN)} \tag{20}$$

The adjusted rand index ranges from -0.5 – for especially discordant clusterings – to 1 – for perfect clusterings. Clusterings with the expected number of correct classifications due to chance have an ARI of 0.

- *Adjusted mutual information* (AMI): uses the notions of entropy and mutual information to measure how much information the clustering gives about the labels. It uses the frequencies of the classes and clusters, which can be collected in a *contingency matrix*, with the rows containing the cluster frequencies per class, and the columns containing the class frequencies per cluster. The mutual information between the classes and clusters can be quantified as

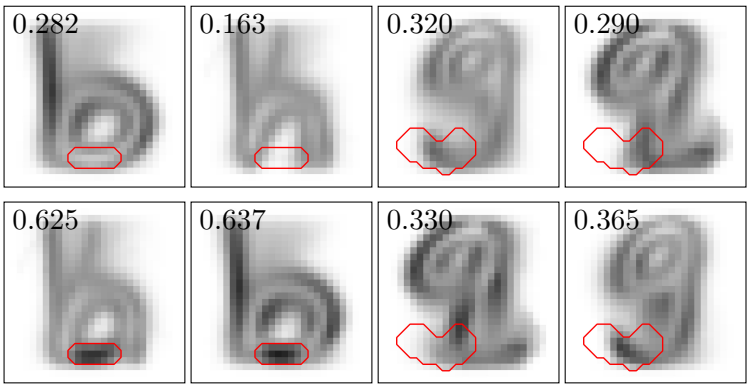
$$\text{MI}(C, L) = \sum_{c}^{|C|} \sum_{l}^{|L|} \frac{|c \cap l|}{n'} \log \frac{|c \cap l|}{|c|\,|l|\,/n'} \tag{21}$$

  Similarly to the rand index, mutual information does not produce a constant value between two random clusterings for different cluster and class sizes. Adjusted mutual information tackles this. It is 1 for perfect agreement, it is a value around 0 for random (independent) clusterings, and can be negative.

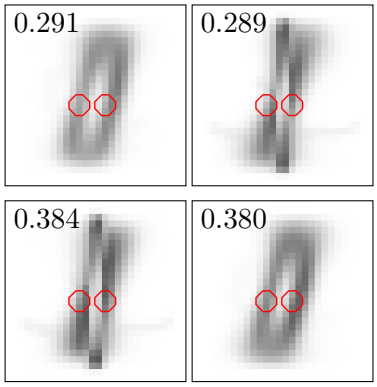
Figure 16 shows how, on average, the difference between a datapoint and the archetype that is supposed to represent its class is less than the difference between it and the archetype of the other class. This means that the majority of datapoints should have their class classified correctly. We see that this is indeed the case in Table 1; even for the most difficult cluster of ***il***, slightly more than half of the datapoints are correctly classified. Note that any random classification has an expected correctness of 50%. In the end, only the correctness of the neighbour relation types matters, as these determine which similarities are removed and which remain. Note that the percentage of correct label classifications anything but determines the percentage of correct relation type classifications, since the latter heavily depends on which labels were correctly classificatied. A label correctness of 0% would actually entail a relation type correctness of 100%, and a label correctness of 50% could still mean a relation type correctness of 0%. However, the two percentages more or less correspond in our results, with relation type correctness ranging from 83% for the ***bh*** cluster to 56% for the ***il*** cluster.

Curiously, the discrepancy between the relation type correctness of interclass neighbours and intraclass neighbours increases significantly for the more difficult clusters. For intraclass neighbours, relation type correctness remains steadily above 70% across all three clusters, while interclass relation type correctness is 79% for the ***bh*** cluster, 30% for the ***gq*** cluster, and 19% for the ***il*** cluster. This means that the separation of all three clusters is rather conservative and that relatively few similarities are removed; few similarities are removed that should not be removed, but also few similarities are removed that should be removed. Despite this, the method does manage to split each cluster into two, leading us to believe that the similarities that are removed are those between neighbours that lie centrally in each cluster's kNN graph.

As to the quality of the separations, still only the ***bh*** cluster shows a satisfactory separation. While the quality of separation of the ***gq*** cluster can still be considered better than a random separation, this is only marginally the case for the ***il*** cluster.



(a) ***bh*** (b) ***gq***



(c) ***il***

Figure 16: Average pairwise per-attribute differences between datapoints from the isolated clusters in Figure 10 and archetypes. For each of the three clusters, we show four visualization for all pairs of ground truth classes and archetypes. The top two visualizations each show the average difference between datapoints from one of the two classes and the respective archetype that represents that class, while the bottom two visualizations each show the difference between datapoint from a classes and the archetype that represents the other class. Ideally, the difference in the top rows are less then the difference in the bottom right, in which case the majority of datapoints would be classified correctly. This is actually the case for all three clusters, which is an explanation for the resemblance-based method's superiority over the other two methods. The difference is largest for the ***bh*** cluster, less so for the ***il*** cluster, and smallest for the ***gq*** cluster.