WeakFlow: Iterative Invertible Distribution Transformations via Weak Destructive Flows

David I. Inouye [1] **Pradeep Ravikumar** [2]

Abstract

While adversarial methods and flows have become the primary methods for transforming between two distributions, both methods generally perform global optimization via end-to-end backpropagation, which requires large computational resources, careful parameter tuning, and gradient-based model components. Thus, we propose an alternative novel meta-algorithm for iteratively transforming between distributions by solving a sequence of simpler layer-wise learning problems. In contrast to end-to-end optimization, each layer-wise problem requires less computational resources, has less hyperparameters, and can be solved via optimization methods other than gradient descent (e.g., tree-based algorithms). Our meta-algorithm iteratively attempts to minimize the mutual information between the distribution indicator label and the latent representation—i.e., destroying (or deconstructing) the differences between distributions. We develop three classes of weak algorithms for the inner optimization of our meta-algorithm: (1) adversarial-based destructors, (2) density destructors, and (3) classifier destructors—a novel class we introduce. We leverage insights from a novel decomposition and optimal transport to develop concrete classifier destructors based on (pseudo-)generative models that deconstruct differences while preserving shared structure as exemplified by the Wasserstein barycenter distribution.

1. Introduction and Motivation

Transforming between two distributions is a fundamental problem in machine learning with applications in generative models and domain translation (among others). Optimal transport (OT) based on the Wasserstein distribution distance has recently become a more widely used tool in ML for theoretically grounded distribution transformations (e.g., see (Peyré & Cuturi, 2019)). The Wasserstein barycenter distribution is a more natural "average" distribution between two (or more) distributions than an additive mixture distribution. However, estimating the OT invertible transformation to its barycenter for continuous space is generally very challenging. Thus, in practice, adversarial methods have become the workhorse for the problem of distribution transformation (e.g., CycleGAN (Zhu et al., 2017)). But adversarial methods often lack exact invertibility or a shared latent space and are known to be challenging and unstable in certain situations. As an alternative to adversarial optimization, normalizing flows (i.e., invertible generative models) can produce exact invertibility by construction and more stable training (Germain et al., 2015; Dinh et al., 2015; Graves, 2016; Dinh et al., 2017; Papamakarios et al., 2017; Inouye & Ravikumar, 2018). Other recent work has even combined adversarial objectives and invertible models to create a hybrid approach between adversarial and flow-based methods (Grover et al., 2020). Yet, both adversarial and flow-based methods usually apply end-to-end backpropagation of large deep models to optimize a global objective function. This end-to-end optimization requires large computational resources, careful parameter tuning, and gradient-based model components. Additionally, the optimization requires backward synchronization between layers—thereby preventing simple pipelining across multiple processors or devices.

Thus, we seek a fundamental alternative to end-to-end backpropagation for transforming between two distributions. We propose a destructive meta-algorithm that seeks to minimize the mutual information (i.e., destroying information) between the latent representation and the domain label by solving a sequence of simpler layer-wise optimization problems, i.e., weak algorithms. In contrast to end-to-end optimization, each layer-wise problem requires less computational resources, has less hyperparameters, and can be solved via optimization methods other than gradient descent (e.g., tree-based algorithms). Our key contributions can be summarized as follows:

- We propose a destructive meta-algorithm for iteratively

[1]Purdue University, West Lafayette, IN, USA [2]Carnegie Mellon University, Pittsburgh, PA, USA. Correspondence to: David I. Inouye <dinouye@purdue.edu>.

Second workshop on *Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models* (ICML 2020), Virtual Conference

transforming between two distributions by minimizing mutual information (i.e., removing the differences between distributions).
- We present three classes of weak algorithms for the inner optimization of our meta-algorithm: (1) density destructors, (2) adversarial-based destructors, and (3) classifier destructors—a novel class we introduce.
- We propose a novel classifier destructors weak algorithm that maps classifiers to invertible transformation by solving local Wasserstein barycenter problems that seek to remove the differences but maintain the similarities between distributions.
- We prove a novel decomposition of mutual information to motivate our classifier destructors algorithm.
- We develop concrete instantiations of our classifier destructors algorithm by leveraging classical classifiers including naïve Bayes, Gaussian Bayes, single index models, and decision tree classifiers.

Notation T or t will denote strong (i.e., deep) or weak invertible transformations respectively. Similarly, D or d will denote strong or weak density destructors (Inouye & Ravikumar, 2018) (described in related work below). We will denote the distribution indicator random variable as $Y \in \{1, 2\}$ and the corresponding data random variables as X_1 and X_2 respectively. The symbol $\circ$ will denote function composition, e.g., $f \circ g = f(g(\cdot))$.

Related Work The closest work to our is the iterative Gaussianization methods for learning density models (Tabak & Vanden-Eijnden, 2010; Tabak & Turner, 2013; Chen & Gopinath, 2000; Lin et al., 2000; Lyu & Simoncelli, 2009; Laparra et al., 2011; Ballé et al., 2016), which all leverage an iterative procedurer by solving a sequence of simpler problems. Density destructors (Inouye & Ravikumar, 2018) are similar in concept to Gaussianization. Inouye & Ravikumar (2018) define a density destructor as an invertible function that transforms a given density to the uniform density (the assumed latent distribution). Given this, Inouye & Ravikumar (2018) provide a meta-algorithm for building up deep density models by iteratively solving weak density estimation problems (e.g., Gaussian mixture, independent component densities, tree densities) and then mapping these densities to density destructors. Because the latent distribution is assumed to be uniform instead of Gaussian (as in most normalizing flow models), the total log likelihood is simply the sum of log likelihoods of all layers, i.e., $\log P_X(\boldsymbol{x}) = \log P_Z(D(\boldsymbol{x}))|J_D(\boldsymbol{x})| = \log |J_D(\boldsymbol{x})| = \sum_{\ell=1}^{k} \log |J_{d^{(\ell)}}(\boldsymbol{x}^{(\ell)})|$, where $D = d^{(k)}(\cdots d^{(1)}(\boldsymbol{x}))$ is a deep density destructor with weak density destructors $d^{(\ell)}$, $\boldsymbol{x}^{(\ell)} = d^{(\ell-1)}(\cdots d^{(1)}(\boldsymbol{x}))$ and $|J_D(\boldsymbol{x})|$ denotes Jacobian determinant of the invertible transformation D. Thus invertible function estimation can be simplified into two steps: 1) standard density estimation (e.g., $\hat{P} = \arg\min_P \text{MLE}(P, \mathcal{X})$ where $\mathcal{X}$ is a set of data samples) and 2) mapping of the estimated density to a corresponding density destructor, i.e., $\hat{D} \triangleq \text{DensDestructor}(\hat{P})$ such that $|J_{\hat{D}}(\boldsymbol{x})| = \hat{P}(\boldsymbol{x})$ where DensDestructor is a deterministic mapping from the density parameters to an invertible transformation. (Inouye & Ravikumar, 2018) generalize previous density destructor mappings from the iterative Gaussianization literature as well as introduce novel ones including the tree density destructors. This weak algorithm can be applied multiple times using the output of the previous transformation as the input to the next iteration to create a deep density destructors (Inouye & Ravikumar, 2018). While we are inspired by iterative Gaussianization and density destructors, our task is fundamentally more challenging because we seek to transform between any two arbitrary distributions while iterative Gaussianization assumes that one of the distributions is Gaussian.

2. WeakFlow: Iterative Destructive Learning

Objective: Minimize mutual information in latent space The overall destructive learning objective is to minimize the mutual information between the indicator random variable Y and the latent random variable Z (which could be dependent on Y):

$$\min_{T_1,T_2} I(Z,Y), \text{ where } Z \triangleq \begin{cases} T_1(X_1), & \text{if } Y = 1 \\ T_2(X_2), & \text{if } Y = 2 \end{cases}.$$

Intuitively, this means we seek to destroy any information that the random variable Z contains about Y and vice versa. When mutual information is zero, then the two random variables are independent—which corresponds to the two distributions P_{Z_1} and P_{Z_2} aligning perfectly, i.e., $P_{Z_1} = P_{Z_2} = P_Z$.

Equivalence to adversarial objective We derive the min-max adversarial objective using simple known identities between mutual information, Jensen-Shannon divergence, and adversarial learning: $I(Z,Y) = \text{JSD}(P_{Z_1}, P_{Z_2}) = \frac{1}{2} \max_f \mathbb{E}_{Z_1}[\log f(\boldsymbol{z})] + \mathbb{E}_{Z_2}[\log(1 - f(\boldsymbol{z}))]$, where $f \colon \mathcal{Z} \to [0, 1]$ is an arbitrary function with output between 0 and 1. Thus, a new min-max adversarial optimization problem can be formulated as follows:

$$\min_{T_1,T_2} \max_f \underbrace{\mathbb{E}_{Z_1}[\log f(\boldsymbol{z})] + \mathbb{E}_{Z_2}[\log(1 - f(\boldsymbol{z}))]}_{\text{Negative classification CE loss in } \textbf{latent} \text{ space}}, \quad (1)$$

where $Z_1 = T_1(X_1), Z_2 = T_2(X_2)$. However, this adversarial optimization has two fundamental issues. First, for the equivalence with JSD, the function f must have arbitrary complexity and the expectations must be at the population level rather than empirical expectations—this is a particular issue for our setup in which we want to solve simple local problems. Second, adversarial optimization is known to be

unstable in many cases. Thus, we propose a novel different decomposition which inspires the novel classifier destructors algorithm described later (proofs in the appendix).

Theorem 1 (Observed Space Decomposition)**.** *If $T_1, T_2 \colon \mathcal{X}^m \to [0,1]^m$ are invertible transformations and we define the following auxiliary function g as: $g(\boldsymbol{x}; T_1, T_2) \triangleq \frac{|J_{T_1}(\boldsymbol{x})|}{|J_{T_1}(\boldsymbol{x})|+|J_{T_2}(\boldsymbol{x})|}$, we can decompose the mutual information in the latent space as follows:*

$$
\begin{aligned}
I(Z,Y) = & \underbrace{JSD(P_{X_1}, P_{X_2})}_{\text{Constant w.r.t. } T_1 \text{ and } T_2} \\
& + \underbrace{\tfrac{1}{2}(\mathbb{E}_{X_1}[-\log g(\boldsymbol{x}; T_1, T_2)] + \mathbb{E}_{X_2}[-\log(1 - g(\boldsymbol{x}; T_1, T_2))])}_{\text{Classification CE loss in } \textbf{observed} \text{ space}} \\
& + \underbrace{\mathrm{KL}(\tfrac{1}{2}(P_{X_1} + P_{X_2}), \tfrac{1}{2}(|J_{T_1}| + |J_{T_2}|))}_{\text{Observed mixture vs Jacobian mixture}} \\
& + \underbrace{H(\tfrac{1}{2}(P_{Z_1} + P_{Z_2}))}_{\text{Entropy of latent mixture}} - \log 2 \,. \qquad (2)
\end{aligned}
$$

Remark 1 (Classification loss in observed space)**.** While this decomposition also reveals a classification problem (as in the adversarial decomposition), it is critical that the classification loss in Theorem 1 is in the *observed space*, i.e., in the space of X_1 and X_2—in contrast to the adversarial loss of Eqn. 1 which is in the latent space Z_1 and Z_2. This observation will be our primary inspiration for our classifier destructors weak algorithm. While our approach does not explicitly account for the other terms (i.e., the KL term and entropy term), we find empirically that our method does decrease the mutual information.

Meta-algorithm: Iterative and destructive compositional learning We now present our iterative meta-algorithm in Alg. 1 for minimizing the mutual information of the transformed representation by iteratively composing simple invertible transformations. The weak algorithm proceeds in a forward-only manner; thus, during training, we only require storing the current representation in memory. Also, this meta-algorithm is amenable to online learning and pipelined learning because of its forward-only manner.

Algorithm 1 Iterative Compositional Meta-Algorithm

Input: Samples from P_{X_1} and P_{X_2} denoted as $\mathcal{X}_1$ and $\mathcal{X}_2$, number of iterations/layers M
Output: Invertible transformations T_1, T_2

$\mathcal{Z}_1^{(0)} \leftarrow \mathcal{X}_1, \quad \mathcal{Z}_2^{(0)} \leftarrow \mathcal{X}_2$
for $\ell = \{1, 2, \ldots, M\}$ **do**
  $t_1^{(\ell)}, t_2^{(\ell)} \leftarrow \texttt{WeakAlgorithm}(\mathcal{Z}_1^{(\ell-1)}, \mathcal{Z}_2^{(\ell-1)})$
  $\mathcal{Z}_1^{(\ell)} \leftarrow t_1^{(\ell)}(\mathcal{Z}_1^{(\ell-1)}), \quad \mathcal{Z}_2^{(\ell)} \leftarrow t_2^{(\ell)}(\mathcal{Z}_2^{(\ell-1)})$
  $T_1^{(\ell)} \leftarrow t_1^{(\ell)} \circ T_1^{(\ell-1)}, \quad T_2^{(\ell)} \leftarrow t_2^{(\ell)} \circ T_2^{(\ell-1)}$
end for
return $T_1^{(M)}, T_2^{(M)}$

3. Weak Algorithms

Adversarial Weak Algorithm Given the equivalence of minimizing mutual information to an adversarial objective, we could merely solve small adversarial optimization problems for invertible transformations: $\arg\min_{t_1,t_2} \max_f \left(\widehat{\mathbb{E}}_{t_1(X_1)}[\log f(\boldsymbol{z})] + \widehat{\mathbb{E}}_{t_2(X_2)}[\log(1 - f(\boldsymbol{z}))]\right)$. However, we do not pursue this further because of the instability of adversarial optimization.

Density Destructors Weak Algorithm (Inouye & Ravikumar, 2018) The simplest non-adversarial algorithm that we will use as a baseline is merely to estimate density destructors (Inouye & Ravikumar, 2018) for each distribution independently: $\forall y \in \{1,2\}, \widehat{P}_{X_y} \leftarrow \arg\min_Q \mathrm{MLE}(Q; \mathcal{X}_y)$, $t_y \equiv d_y \leftarrow \mathrm{DensDestructor}(\widehat{P}_{X_y})$, where $\mathcal{X}_1$ and $\mathcal{X}_2$ are samples from P_{X_1} and P_{X_2} respectively. Given a flexible enough density destructor (or normalizing flow), both distributions will converge to the uniform distribution—and thus the mutual information will be zero, i.e., $\lim_{\ell \to \infty} I(Z^{(\ell)}, Y) = 0$.

Classifier Destructors Weak Algorithm Finally, we propose a novel classifier destructor algorithm inspired by the observed classification loss in Theorem 1. We focus on minimizing the second term of Theorem 1 because that can be framed as a standard classification problem in the observed space, i.e., we won't have to do adversarial learning. Our general approach will be to map the classifier to a simple (pseudo-)generative classifier with (pseudo-)densities Q_1 and Q_2, and then use the machinery of density destructors (Inouye & Ravikumar, 2018) to map these densities to invertible transformations. Additionally, unlike in density destructors that move the distribution to a high entropy distribution, we seek to maintain the shared structure between distributions. Thus, we consider using local Wasserstein barycenter projections in our algorithms to preserve as much structure as possible while hoping to make the distributions overlap. (We review a few well-known results about Wasserstein barycenters in the appendix.) We define several properties in the appendix about classifier destructors but jump to our concrete instantations for this extended abstract:

Proposition 2 (Naïve Bayes Classifier Destructors)**.** *Given a naïve Bayes classifier with $P_Y(1) = P_Y(2) = \frac{1}{2}$: $f(\boldsymbol{x}) = \frac{\prod_{i=1}^m \widehat{P}_{X_{1,i}}(x_i)}{\prod_{i=1}^m \widehat{P}_{X_{1,i}}(x_i) + \prod_{i=1}^m \widehat{P}_{X_{2,i}}(x_i)}$, the following feature-wise composite transformations are Wasserstein classifier destructors: $t_y(\boldsymbol{x}) = [d^{-1}_{\text{bary},1}(d_{y,1}(x_1)),\ d^{-1}_{\text{bary},2}(d_{y,2}(x_2)), \cdots, d^{-1}_{\text{bary},m}(d_{y,m}(x_m))]$, where $d_{y,i} = \mathrm{DensDestructor}(\widehat{P}_{X_{y,i}})$, $d^{-1}_{\text{bary},i} = \mathrm{DensDestructor}(\mathrm{bary}(\widehat{P}_{X_{1,i}}, \widehat{P}_{X_{2,i}}))$, and $\mathrm{bary}(\cdot,\cdot)$ denotes the Wasserstein barycenter distribution.*

Proposition 3 (Gaussian Bayes Classifier Destruc-

tors). *Given a Gaussian Bayes classifier:* $f(\boldsymbol{x}) = \frac{P_{\mathcal{N}(\mu_1,\Sigma_1)}(\boldsymbol{x})}{P_{\mathcal{N}(\mu_1,\Sigma_1)}(\boldsymbol{x})+P_{\mathcal{N}(\mu_2,\Sigma_2)}(\boldsymbol{x})}$, *the following composite transformations are Wasserstein classifier destructors:* $t_y = d_{\text{bary}}^{-1} \circ d_y$, *where* $d_y(\boldsymbol{x}) \triangleq \Phi(A_y(\boldsymbol{x} - \mu_y))$ *and* $d_{\text{bary}}(\boldsymbol{x}) \triangleq \Phi(\Sigma_{\text{bary}}^{-1/2}(\boldsymbol{x} - \mu_{\text{bary}}))$ *are density destructors for* $P_{\mathcal{N}(\mu_y,\Sigma_y)}$ *and* $P_{\mathcal{N}(\mu_{\text{bary}},\Sigma_{\text{bary}})}$ *respectively, and* $A_y \triangleq \Sigma_{\text{bary}}^{-1/2}\frac{1}{2}(I + B_y)$ *where* $B_y \triangleq \Sigma_y^{-1/2}(\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2}\Sigma_y^{-1/2}$ *is the projection matrix for the optimal Monge map between the two class distributions (if* $y = 1$*, then* $\tilde{y} = 2$ *or vice versa).*

Leveraging discriminative classifiers to find structure Now we consider how we can leverage discriminative classifiers to create pseudo-generative classifiers that focus more on the differences between distributions. First, we extend naïve Bayes classifier destructors by finding an orthogonal pre-transformation that exposes the linear directions that have high mutual information (i.e., directions that will give good classification)—one special case is a random orthogonal pre-transformation. Specifically, we designed a method that optimizes k *single index models* along k orthogonal directions simultaneously, where a *single index model* (SI) is a model that only operates along one direction (i.e., a single index): $f_{\text{SI}}(\boldsymbol{x}) \triangleq g(\beta^T \boldsymbol{x})$ where β is a vector and $g\colon \mathbb{R} \mapsto \mathbb{R}$ is a arbitrary scalar function (e.g., a deep network). Thus, jointly estimating k SI models can be seen as estimating a single index ensemble (SIE) (more details in appendix). The learned orthogonal transformation is then used for pre and post processing transformations around naïve Bayes classifier destructors to significantly increase the modeling power, denoted as SIE NB in the preliminary experiments. As another example, we can leverage decision tree classifiers $f_{S,\mathcal{L}}(\boldsymbol{x}_i)$ (where S is the tree split structure and $\mathcal{L}$ are the classifier probabilities at the leaves) to expose discriminative structure. We can use tree density destructors developed in (Inouye & Ravikumar, 2018) to create a novel tree-based approach, which is impossible with gradient-based optimization methods as used in standard end-to-end learning (more details in appendix).

4. Experiments

Simulated Experiments We first begin with some simulated experiments to develop some intuition about classifier destructors and to compare to density destructors (experimental details in appendix). The progression through various stages can be seen in Fig. 1 for the moons dataset. To estimate the decrease in mutual information, we fit Gaussian process classifiers with default parameters from scikit-learn for these datasets (using separate train and test datasets) and take the test accuracy as a proxy for mutual information. Additionally, we measure the distance of the transformed representation from its original representation and compare to the optimal empirical Barycenter mapping—this helps quantify if the structure is preserved. The empirical optimal transport distance to the barycenter is also shown as a dashed line—demonstrating that our classifier destructors can sometimes achieve close to the optimal barycenter transformation. MI proxy and distance can be seen in Fig. 2.

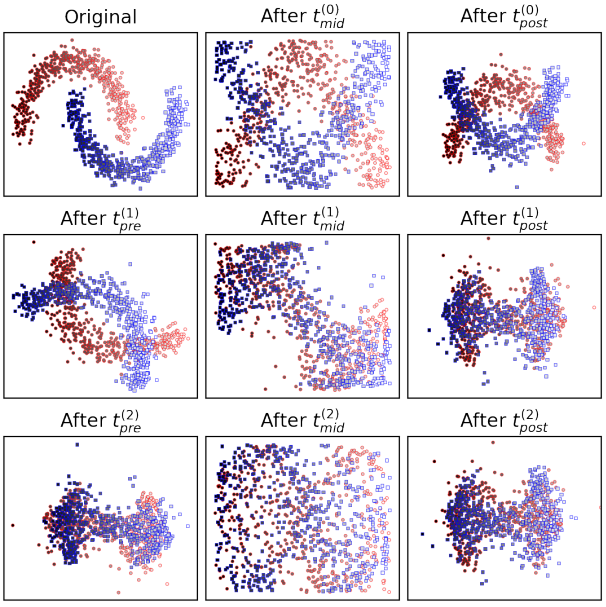

Figure 1: Our proposed method SIE NB aligns the distributions, denoted by color, in only *two* layers while density destructors (in appendix) takes seven or more layers to align the distributions (superscripts denote layer number and "pre" and "post" are optional pre and post processing transforms while "mid" is the core transformation). Additionally, SIE NB preserves some of the original structure (e.g., the final latent distribution has a curved shape as in the original).

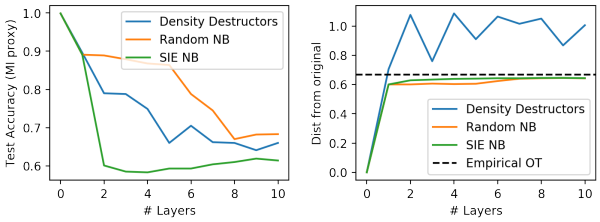

Figure 2: (Left) Preliminary simulated experiments demonstrate that our proposed method (SIE NB) aligns the two distributions (as measured by test accuracy where lower means better aligned) with far fewer layers than the two baselines of density destructors or NB with only random rotations. (Right) Additionally, our proposed method (SIE NB) minimally distorts the original distributions (as measured by the average distance before and after the transformation where lower is better) as evident by the distance approaching the optimal—i.e., minimal—transport (OT) distance (dashed line); however, density destructors significantly distort the distributions more than necessary. More results in appendix.

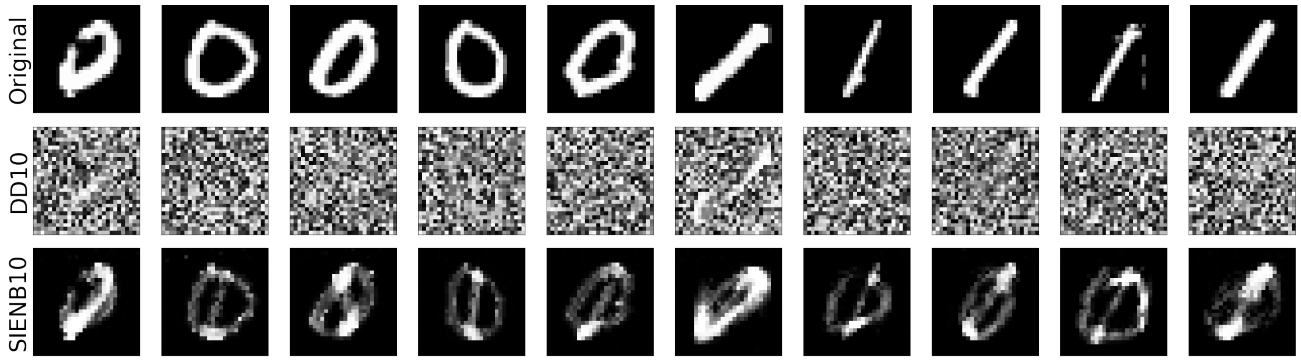

Figure 3: Latent representations after applying various classifier destructors. DD10 is for density destructors with 10 layers. SIENB10 is SIE naïve Bayes with 10 layers. Additional results in appendix.

High-dimensional Experiment To show the feasibility of using our proposed algorithms on high dimensional data, we explore the case of transforming between the 0s and 1s digits on the permuted MNIST dataset (i.e., we do not consider pixel dependencies, other experimental details in appendix). The results in Fig. 3 show that our core classifier destructors are able to move the 0 and 1 MNIST distributions much closer to each other than the original space while maintaining reasonable image structure such as the shared black pixels. To measure a proxy for mutual information, we again train a classifier (a simple deep CNN) to see if the images are getting harder to distinguish. The accuracies using a deep CNN classifier when transforming between 0 and 1 and vice versa from top to bottom are 1.000 (Original), 1.000 (DD10), and 0.988 (SIENB10) respectively—showing that our classifier destructors begin to degrade the accuracy (a proxy for mutual information) even on this very simple classification problem.

Acknowledgements

PR acknowledges support from the NSF via IIS-1909816.

References

Ballé, J., Laparra, V., and Simoncelli, E. P. Density modeling of images using a generalized normalization transformation. *ICLR*, pp. 1–12, 2016.

Chen, S. S. S. and Gopinath, R. A. Gaussianization. *NIPS*, pp. 423–429, 2000.

Chen, Y., Georgiou, T. T., and Tannenbaum, A. Optimal transport for gaussian mixture models. *IEEE Access*, 7:6269–6278, 2019.

Dinh, L., Krueger, D., and Bengio, Y. NICE: Non-linear independent components estimation. In *arXiv preprint arXiv:1410.8516*, 2015.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real NVP. In *ICLR*, 2017.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. MADE: Masked autoencoder for distribution estimation. In *ICML*, 2015.

Graves, A. Stochastic backpropagation through mixture density distributions. *arXiv:1607.05690v1*, 2016.

Grover, A., Chute, C., Shu, R., Cao, Z., and Ermon, S. Alignflow: Cycle consistent learning from multiple domains via normalizing flows. In *AAAI*, 2020.

Inouye, D. I. and Ravikumar, P. Deep density destructors. In *International Conference on Machine Learning (ICML)*, pp. 2172–2180, jul 2018.

Laparra, V., Camps-Valls, G., and Malo, J. Iterative Gaussianization: From ICA to random rotations. *IEEE Transactions on Neural Networks*, 22(4):537–549, 2011.

Lin, J., Saito, N., and Levine, R. An iterative nonlinear Gaussianization algorithm for resampling dependent components. *Proc. 2nd International Workshop on Independent Component Analysis and Blind Signal Separation*, pp. 245–250, 2000.

Lyu, S. and Simoncelli, E. P. Nonlinear extraction of independent components of natural images using radial Gaussianization. *Neural computation*, 21:1485–1519, 2009.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *NIPS*, 2017.

Peyré, G. and Cuturi, M. Computational optimal transport. *Foundations and Trends© in Machine Learning*, 11(5-6):355–607, 2019. ISSN 1935-8237. doi: 10.1561/2200000073. URL http://dx.doi.org/10.1561/2200000073.

Tabak, E. G. and Turner, C. V. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

Tabak, E. G. and Vanden-Eijnden, E. Density estimation by dual ascent of the log-likelihood. *Commun. Math. Sci.*, 8(1):217–233, 2010.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

A. Introduction to Previously Known Results about Wasserstein Barycenters

We briefly introduce a few known results about Wasserstein barycenters that will be critical for our development. While solving for barycenter distributions or optimal transportation maps (also known as Monge maps), there are several simple cases that are known in closed-form (particularly 1D distributions and multivariate Gaussian distributions. The optimal transport map for continuous distributions from a distribution denoted by α to β is known as the Monge map, which we will denote by $M^*_{\alpha\to\beta}$. For 1D distributions, the optimal Monge map is merely a composition of CDF functions: $M^*_{\alpha\to\beta} = F^{-1}_\beta \circ F_\alpha$, where F_α and F_β are the univariate CDF functions of α and β respectively. The optimal Monge map from one Gaussian $\mathcal{N}(\mu_1, \Sigma_1)$ to another Gaussian $\mathcal{N}(\mu_2, \Sigma_2)$ is (Peyré & Cuturi, 2019)[Remark 2.31]: $M^*_{1\to2}(\boldsymbol{x}) = A(\boldsymbol{x} - \mu_1) + \mu_2$ where $A = \Sigma_1^{1/2}(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}\Sigma_1^{-1/2}$. Additionally, the optimal interpolation between distributions indexed by $\lambda \in [0, 1]$ is given by the McCann's interpolation (Peyré & Cuturi, 2019)[Remark 7.1]: $M^*_\lambda(\boldsymbol{x}) = (1-\lambda)\boldsymbol{x} + \lambda D_{1\to2}(\boldsymbol{x})$, where $\lambda = 0.5$ corresponds to the mapping to the barycenter distribution. Thus, by combining these two facts, we can derive the optimal transformation from each distribution to the barycenter distribution in the case of 1D distributions or multivariate Gaussian distributions. Additionally, we note that the barycenter of two Gaussian distributions has a mean and covariance as follows (Chen et al., 2019, p. 6271): $\mu_{\text{bary}} = \frac{1}{2}(\mu_2 + \mu_1)$, $\Sigma_{\text{bary}} = \Sigma_1^{-1/2}(\frac{1}{2}(\Sigma_1 + (\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}))^2\Sigma_1^{-1/2}$.

B. Definitions and Properties of Classifier Destructors

We now define several properties of a pair of transformations (t_1, t_2) corresponding to the two distributions.

Definition 1 (Classifier Consistency Property)**.** *Given a classifier f, a pair of transformations (t_1, t_2) is said to be consistent with f, if $\forall \boldsymbol{x}$, $g(\boldsymbol{x}; t_1, t_2) = f(\boldsymbol{x})$, where $g(\boldsymbol{x}; t_1, t_2)$ is defined as in Theorem 1.*

Definition 2 (Wasserstein Barycenter Mapping Property)**.** *Given a (pseudo-)generative classifier f with class distributions Q_1 and Q_2, the transformations t_1 and t_2 are equivalent to the optimal Monge maps between the corresponding class conditional distribution and the Wasserstein barycenter distribution, i.e., $t_1 = M^*_{Q_1\to\text{bary}}$ and $t_2 = M^*_{Q_2\to\text{bary}}$.*

Classifier consistency will ensure that the observed classification error term in Eqn. 2 is reduced at every iteration (formal statement given later in Cor. 4). *Wasserstein barycenter mapping property* will help ensure that similar structure is maintained. Given these definitions, we can now define our primary objects called *classifier destructors* and an extension called *Wasserstein classifier destructors*.

Definition 3 (Classifier Destructors)**.** *Given a probabilistic classifier f,* classifier destructors *are a pair of composite invertible transformations (t_1, t_2) defined as: $t_1 \triangleq t_{\text{post}} \circ t_{\text{mid},1} \circ t_{\text{pre}}$ and $t_2 \triangleq t_{\text{post}} \circ t_{\text{mid},2} \circ t_{\text{pre}}$, where t^{pre} and t^{post} are shared pre and post processing transformations, such that* classifier consistency *(Def. 1) holds for the transformations $(t_{\text{mid},1}, t_{\text{mid},2})$ and the classifier $\tilde{f} = f \circ t_{\text{pre}}$ (equivalent classifier after preprocessing transformation).*

Definition 4 (Wasserstein Classifier Destructors)**.** *Given a (pseudo-)generative classifier f,* Wasserstein classifier destructors *are classifier destructors (t_1, t_2) such that the* Wasserstein barycenter mapping *property (Def. 2) also holds for (t_1, t_2).*

We prove in Cor. 4 that if our classifier is better than random, then the corresponding classifier destructors reduce the classification loss term in Theorem 1. Note that this does not necessarily mean that the mutual information is decreased at every iteration as there are other terms we do not explicitly control for; however, our empirical results suggest that focusing only on this term can still be useful and does not require unstable adversarial optimization. Now we present two concrete instatiations of Wasserstein classifier destructors.

C. Leveraging Single Index Ensembles

In the previous case, we had generative models directly and the key difference from density destructors is the Wasserstein barycenter mapping property. Now we consider how we can leverage discriminative classifiers to create pseudo-generative classifiers that focus more on the differences between distributions—the parts that will reduce classification loss—rather than the similarities. These auxiliary classifiers will only be used to find important structure but can be discarded after training (similar to how discriminative networks in GANs are no longer needed after training). First, we would like to extend naïve Bayes classifier destructors by performing an orthogonal pre-transformation that exposes the dimensions that have high mutual information (i.e., directions that will give good classification). As a first simple idea, we propose to find a single direction that is most interesting using single index classifier models $f(\boldsymbol{x}) = h(\beta^T\boldsymbol{x})$, where h can be an arbitrarily complex univariate function bounded between 0 and 1, and β is the projection direction. We can map this single direction to a an orthogonal transformation via Householder reflectors. In general, a single direction may not provide much information

in high dimensions, so we also consider a single index ensemble (SIE) approach to learn multiple single index classifiers in parallel: $W_{\text{SIE}} = \arg\min_W \min_{h_1 \ldots h_k} \sum_{j=1}^{k} \sum_{i=1}^{n} \mathcal{L}(h_j([W\boldsymbol{x}_i]_j), y_i)$, where $k \leq m$ is the number of single index models to learn, W is an orthogonal matrix (parameterized via k Householder reflectors in our experimetns), and $\mathcal{L}$ is the cross entropy loss. Thus, combining this with naïve Bayes classifiers destructors as pre and post transformations we have that $t_{\text{pre}}(\boldsymbol{x}) = W_{\text{SIE}}\boldsymbol{x}$, $t_{\text{mid},1}, t_{\text{mid},2} = \text{NaiveBayesClassifierDestructors}(Q_1, Q_2)$, and $t_{\text{post}} = W_{\text{SIE}}^{-1}\boldsymbol{x}$, where Q_1 and Q_2 are the independent conditional distributions after the pretransformation. An idea for tree destructors is in the appendix.

D. Leveraging Decision Tree Classifiers

We also propose to leverage decision tree classifiers $f_{S,\mathcal{L}}(\boldsymbol{x}_i)$, where S is the tree split structure and $\mathcal{L}$ are the classifier probabilities at the leaves, to expose discriminative structure. Then, we can estimate tree density destructors from (Inouye & Ravikumar, 2018) for each class distribution given the tree split structure S (we ignore the $\mathcal{L}$ probabilities similar to how we ignore h_j of the SIE model). Given the structure S, this is simply leaf node bin counting to estimate a tree densities. Finally, we can solve local 1D Wasserstein barycenter problems starting from the leaves and bubbling upward (note that at each split, it is like solving a 1D histogram barycenter problem with only two bins in each histogram). We emphasize that our meta-algorithm does not require gradient-based weak algorithms and so we are free to use trees.

E. Experimental Details

Since we will be using classifier accuracy on the transformed data as a proxy for mutual information (MI) between the representations, we will need both train and test splits for this classifier. We will call this classifier used for evaluation the "MI Classifier". Thus, for both experiments, we will split the data into three disjoint splits because we need separate data for evaluation.

1. *Train* ($\mathcal{D}_{\text{train}} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n_{\text{train}}}$) - This first split is the training data used for our meta-algorithm to learn the transformation between domains.
2. *MI Train* ($\mathcal{D}_{\text{MI\_train}} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n_{\text{MI\_train}}}$) - This second split is the data used to train our MI classifier model whose test accuracy will be a proxy for mutual information.
3. *MI Test* ($\mathcal{D}_{\text{MI\_test}} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n_{\text{MI\_test}}}$) - This last split is the data used to test our classifier model whose test accuracy will be a proxy for mutual information.

E.1. Simulated Experiment

We first begin with some simulated experiments to develop some intuition about classifier destructors and to compare to density destructors. We generate the simple moons and classification datasets via scikit-learn. We apply the density destructor weak algorithm using random rotations followed by independent density destructors. We then apply our SIE naïve Bayes weak algorithm (SIENB) to the same dataset. The progression through various stages can be seen in Fig. 1 for the moons dataset. SIENB makes the the distributions overlap quickly (with only 3 full transformations) while maintaining some structural properties and the correspondence of points (denoted by dark vs light across x-axis). Density destructors weak algorithm (right) do not maintain the original structure of the data including correspondence between points. Random rotation naïve Bayes (shown in the appendix) maintains some of the correspondence but does not perform as well. To estimate the decrease in mutual information, we fit Gaussian process classifiers with default parameters from scikit-learn for these datasets (using separate train and test datasets) and take the test accuracy as a proxy for mutual information. Additionally, we measure the distance of the transformed representation from its original representation and compare to the optimal empirical Barycenter mapping—this helps quantify if the structure is preserved. In Fig. 2, we can see that SIE naïve Bayes weak algorithm on the moons (left) and simple classification (middle left) datasets quickly reduces classifier test accuracy on latent representation (a proxy for mutual information) while random naïve Bayes and density destructors are much slower. Both SIE naïve Bayes and random rotation naïve Bayes on the moons (middle right) and blobs (right) only minimally move the data points from their original location while density destructors ignores the original structure or point correspondence. The empirical optimal transport distance to the barycenter is also shown as a dashed line—demonstrating that our classifier destructors can sometimes achieve close to the optimal barycenter transformation. Note that the test classifier used as a proxy for mutual information is trained independently for every latent representation.

Datasets and Setup We generate 3000 2D samples the moons dataset, the circles dataset, and the classification dataset via the sklearn functions `make_moons`, `make_circles` and `make_classification`. We split this into our three subsets

above so that $n_{\text{train}} = n_{\text{MI\_train}} = n_{\text{MI\_test}} = 1000$. For our MI classifier, we first normalize the data to have zero mean and unit variance and then fit a Gaussian process classifier from sklearn with default parameters using the MI train dataset $\mathcal{D}_{\text{MI\_train}}$. We estimate test accuracy using the held-out MI test dataset $\mathcal{D}_{\text{MI\_test}}$. We compute the average Euclidean distance between the original data and the data in the latent representation space based on $\mathcal{D}_{\text{MI\_train}}$. The OT distance is computed numerically as half distance between the optimal transport between samples of X_1 and X_2 in $\mathcal{D}_{\text{MI\_train}}$. Since the number of samples is not always equal because of randomness, we take the minimum number of samples between $y = 1$ and $y = 2$ so that the empirical OT is an exact matching between samples. Thus, after matching, this merely the distance between samples divided by two (since we project to the barycenter).

Density destructors For the density destructors model, we first apply an independent Gaussian density destructor as an initial destructor to quickly normalize the data with respect to the mean and variance along each dimension. Then, we applied a Gaussian-copula density destructor, which has the form

$$t_{\text{mid},y} = F_{\text{hist},y} \circ \Phi \circ W_y \circ \Phi^{-1}\,, \tag{3}$$

where Φ and Φ^{-1} are the element-wise standard normal CDF and inverse CDF, W_y is a random orthogonal matrix, and $F_{\text{hist},y}$ is a histogram CDF estimated with 100 bins, bounds of 0 to 1, and an regularization parameter $\alpha = 1$ (which are like the pseudo-counts in each bin). This was primarily inspired by the structure of density destructors used in the examples in (Inouye & Ravikumar, 2018).

Naïve Bayes classifier destructors Our naïve Bayes classifier destructors estimate the density of each feature using a very simple density destructor:

$$d_{\text{naïve},i}(\boldsymbol{x}) = F_{\text{hist}}(\Phi(\frac{\boldsymbol{x} - \mu}{\sigma}))\,, \tag{4}$$

where $\Phi(\frac{\boldsymbol{x}-\mu}{\sigma})$ is the density destructor of a normal distribution with mean μ and standard deviation σ, and F_{hist} is a histogram CDF estimated with $\lfloor\sqrt{n_{\text{train}}}\rfloor$ bins (a simple heuristic), bounds from 0 to 1, and an $\alpha = 10^{-6}$ (regularization based on pseudo-counts). The Gaussian preprocessing step projects the data from $\mathbb{R}$ to $[0, 1]$, which is much more amenable to non-parametric histogram density estimation since the data is now bounded—thereby avoiding the need to estimate the support of the histogram. To compute the barycenter projection, we numerically estimate the projection in 1D using 100 grid points via McCann's interpolation and the closed-form solution to 1D optimal transport based on the CDFs of the two densities.

Random rotation naïve Bayes For random rotation naïve Bayes, we first applied a naïve Bayes destructor defined as above. Then, for every future layer, we apply in sequence: a random orthogonal transformation W, a naïve Bayes classifier destructor as defined above, and then the inverse of the pre-rotation $W^{-1} = W^T$.

SIE naïve Bayes For SIE naïve Bayes, we do the same thing as random rotation naïve Bayes except we learn the rotation using a SIE model. For the SIE model, we parameterize the orthogonal matrix W by $k = m$ Householder reflectors (while one Householder reflector would be enough here, in the high-dimensional experiments we will choose $k < m$). Then, for each single index model, we use a deep fully connected model with 2 hidden layers of width 100 and leaky relu activation (using default parameters in PyTorch) with a sigmoid activation at the output. Thus, the input is one dimension, followed by two 100 dimensional hidden layers and then ending with a single dimensional output. We use cross entropy loss and the Adam optimizer with default parameters in PyTorch. We train the SIE model with a batch size of 500 and for 1000 batches (or steps).

Expanded figures Expanded figures for all datasets can be found in the following figures.

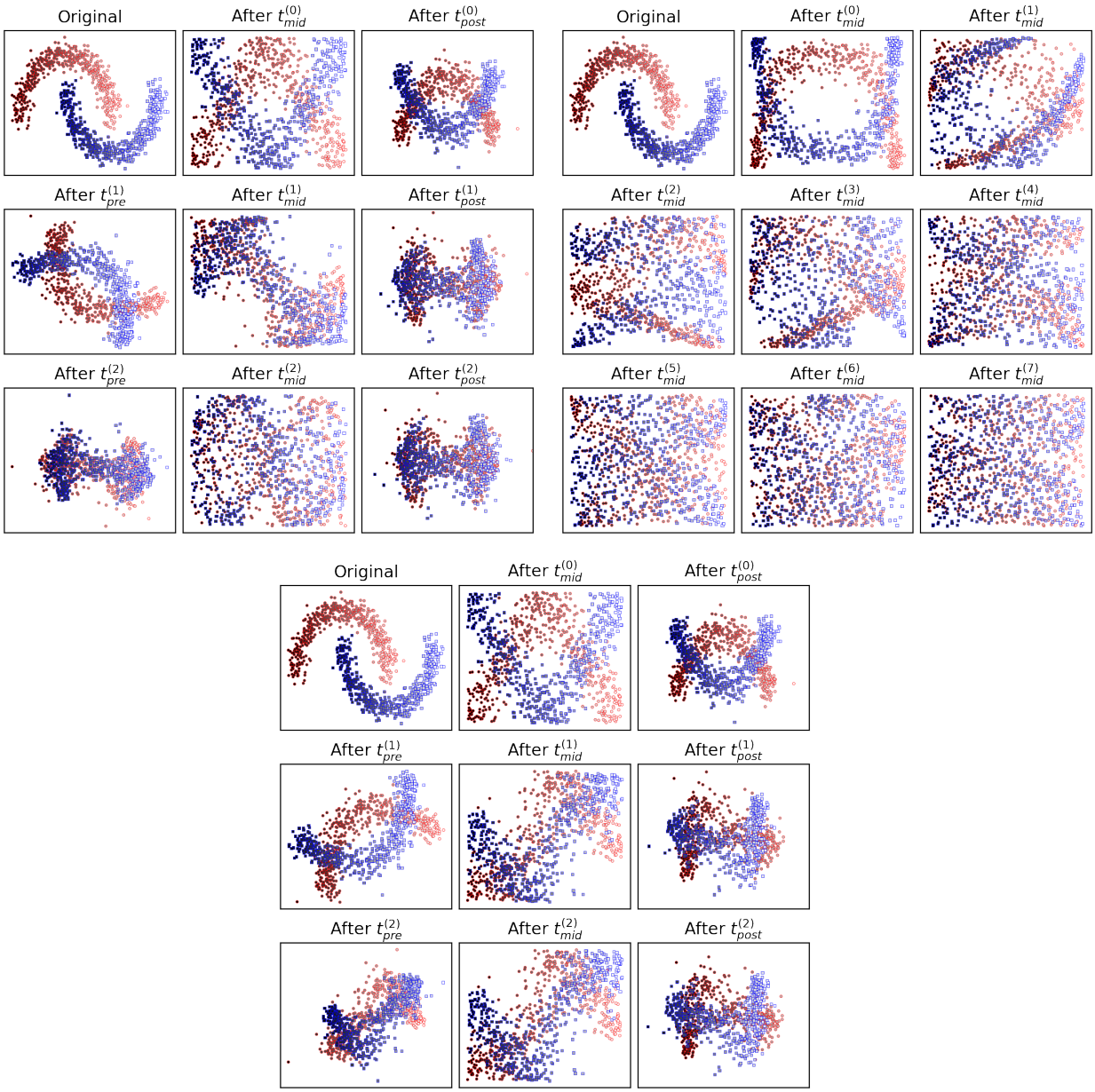

Figure 4: Progression of SIE naïve Bayes (top left), density destructors (top right), and random rotation naïve Bayes (bottom).

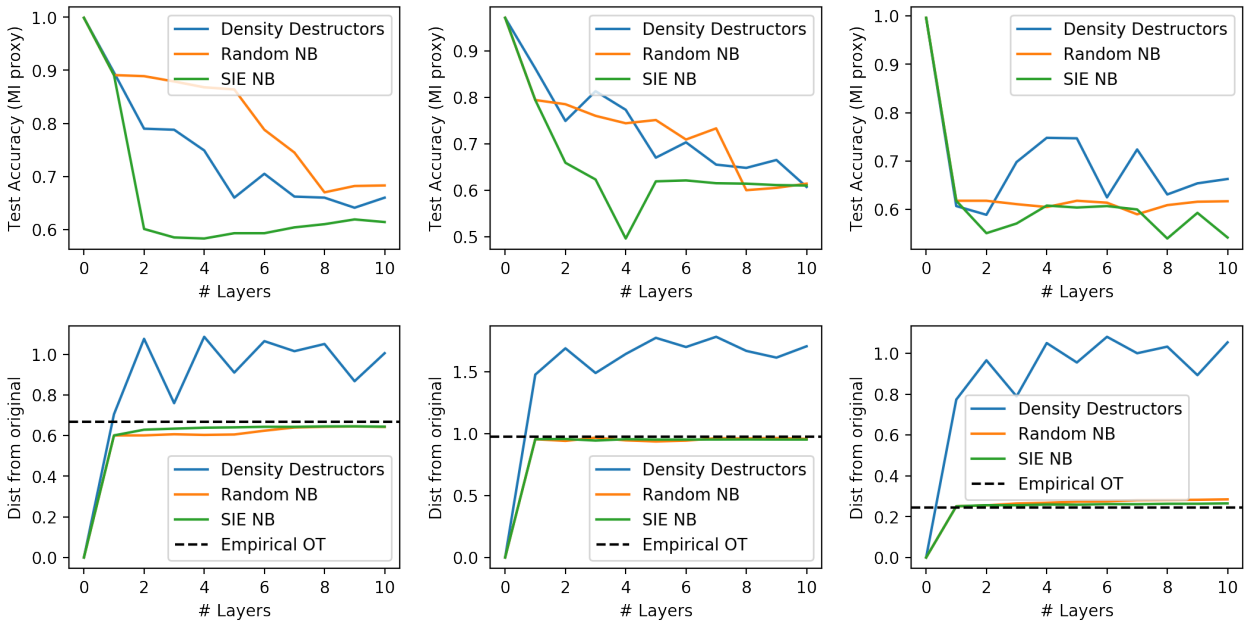

Figure 5: Mutual information at different layers (top) and distance from original representation (bottom). Moons (left), classification (middle) and circles (right).

E.2. High-dimensional Experiment (MNIST)

To show the feasibility of using our proposed algorithms on high dimensional data, we explore the case of transforming between the 0s and 1s digits on the permuted MNIST dataset (i.e., we do not consider image-based dependencies in our methods but merely use MNIST as an example of more general high dimensional problems). We compare a single naïve Bayes classifier destructor (NB), a single Gaussian Bayes classifier destructor (GB), and an deep SIENB(L) classifier destructor with L layers and $k = 100$ single index models per SIE after applying GB and NB pretransformations. The results in Fig. 3 show that our core classifier destructors are able to move the 0 and 1 MNIST distributions much closer to each other than the original space while maintaining reasonable image structure such as the shared black pixels. To measure a proxy for mutual information we again train a classifier (a simple deep CNN) to see if the images are getting harder to distinguish. The accuracies using a deep CNN classifier for these representations from top to bottom are 1.000, 1.000, 0.986, 0.910, and 0.870 respectively—showing that our classifier destructors begin to degrade the discriminator accuracy (a proxy for mutual information) even on this simple classification problem. Additionally, because simple NB does not degrade the accuracy, it seems that the dependencies between pixels is key in the classification task as expected.

Datasets and Setup We took the 0 and 1 digits from the MNIST dataset ($m = 784$). As preprocessing, we added uniform noise of size to convert from discrete to continuous data, i.e.,

$$x' = x + \epsilon/256 \tag{5}$$

where $\epsilon \sim \text{Uniform}([0, 1])$. We then split this into our three subsets above so that $n_{\text{train}} = 6890$, $n_{\text{MI\_train}} = 6890$, and $n_{\text{MI\_test}} = 1000$. For our MI classifier, a deep CNN classifier defined in PyTorch as follows:

```
nn.Sequential(
    # input is (nc) x 28 x 28
    nn.Conv2d(nc, ndf, kernel_size=4, stride=2, padding=1, bias=False),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf) x 14 x 14
    nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
    nn.BatchNorm2d(ndf * 2),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf*2) x 7 x 7
    nn.Conv2d(ndf * 2, ndf * 4, 3, 1, 1, bias=False),
    nn.BatchNorm2d(ndf * 4),
    nn.LeakyReLU(0.2, inplace=True),
    # state size. (ndf*4) x 7 x 7
    nn.Conv2d(ndf * 4, 1, 7, 1, 0, bias=False),
    nn.Sigmoid()
    # state size. 1 x 1 x 1 (use .view(-1) to get result as single array)
)
```

Additionally we initialize the convolutional weights with a zero mean Gaussian with 0.02 standard deviation and the batchnorm layers were initialized with a mean of 1 and a standard deviation of 0.02. These choices were based on adapting the DCGAN PyTorch tutorial at https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html, accessed in June 2020. We estimate test accuracy using the held-out MI test dataset $\mathcal{D}_{\text{MI\_test}}$. Because we use a convolutional classifier, we perform two ways of comparing methods using accuracy of this MI classifier as a proxy.

1. *0 vs 1 in latent space:* This is training and testing the classifier after projecting the MI training and testing datasets into the latent space—i.e., after projecting, these samples should be close. The primary issue with this is that for comparing to density destructors (which do not preserve latent structure), the convolutional image-basic structure no longer exists—thus, a CNN is unlikely to do well since the latent space no longer looks like real images.
2. *Average of real vs fake:* For a more fair comparison to density destructors, we train the CNN classifier to distinguish between the original 0s (or 1s) and 1s that have been projected to the 0s domain by first projecting to the shared latent space via t_1 and then inverse transforming to the 0's space via t_2^{-1}. This essentially creates "fake" 0s (or "fake" 1s). Note that the mutual information is the same in the latent space as it is in the 0's space (or the 1's space) so this is a valid equivalent comparison, i.e., $I(Z, Y) = I(X', Y)$ where $X' = t_1^{-1}(Z)$ so that if $Y = 1$, then

$X' = t_1^{-1}(Z) = t_1^{-1}(t_1(X)) = X$ but if $Y = 2$, then $X' = t_1^{-1}(Z) = t_1^{-1}(t_2(X))$. This metric is more comparable for all methods including density destructors that do not preserve any image-like structure in the latent space.

We compute the average Euclidean distance between the original data and the data in the latent representation space based on $\mathcal{D}_{\text{MI\_train}}$.

Density destructors For our density destructors, we follow the basic structure from the MNIST example given for density destructors in the official density destructors GitHub repository: https://github.com/davidinouye/destructive-deep-learning/blob/master/notebooks/demo_mnist_deep_copula.ipynb. Following this, we have apply an initial histogram density destructor with 256 bins and bounds of 0 and 1 with an $\alpha = 1$. We then use a Gaussian copula destructor for each layer afterwards that has the following form:

$$t_y = \Phi \circ \Omega_{\text{ZCA}} \circ \Phi^{-1} \circ F_{\text{hist}} \tag{6}$$

where Ω_{ZCA} is a ZCA whitening transformation, the histogram used to fit F_{hist} has 40 bins, bounds of 0 and 1 and an $\alpha = 1$, and Φ and Φ^{-1} are defined as before. Each of these are estimated sequentially via density destructors.

Naïve Bayes classifier destructors We use the same settings as in the simulated experiment.

SIE Naïve Bayes classifier destructors We use the same setup as in the simulated experiment except that we set the number of single index classifiers to $k = 100$.

Expanded figures Expanded and additional result figures can be seen in the following figures.

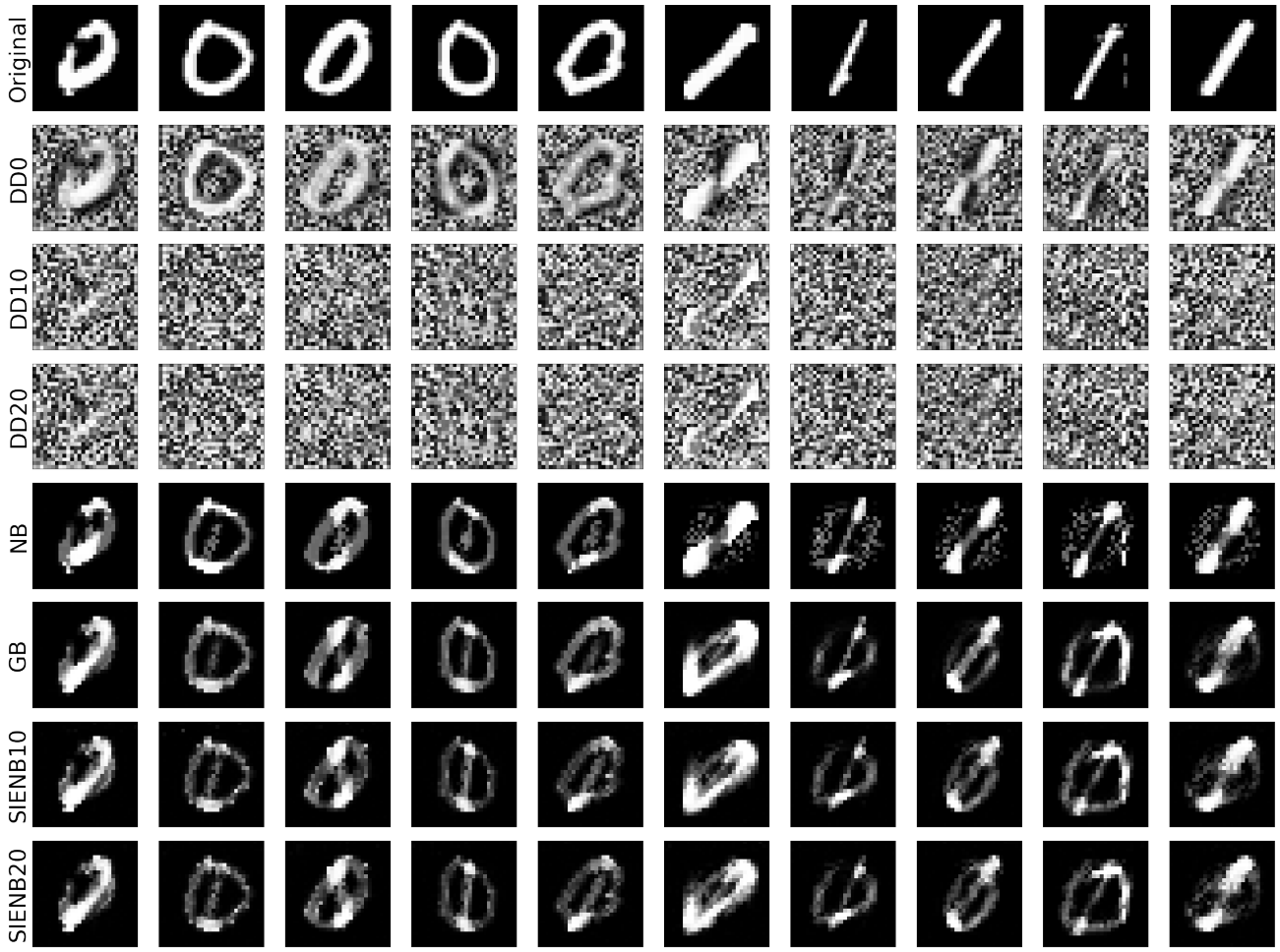

Figure 6: Latent representations after applying various transformations. Notice that density destructors (DD) do not maintain shared image structure since they seek to push the representation toward the uniform distribution everywhere. DD stands for density destructors weak algorithm. DD0 is the initial transformation and DD10 and DD20 are after 10 or 20 layers after the initial transformation. NB stands for naïve Bayes classifier destructors. GB stands for Gaussian Bayes classifier destructors. SIENB10 and SIENB20 are SIE naïve Bayes classifier destructors after 10 and 20 layers respectively (with an initial transformation of GB followed by NB).

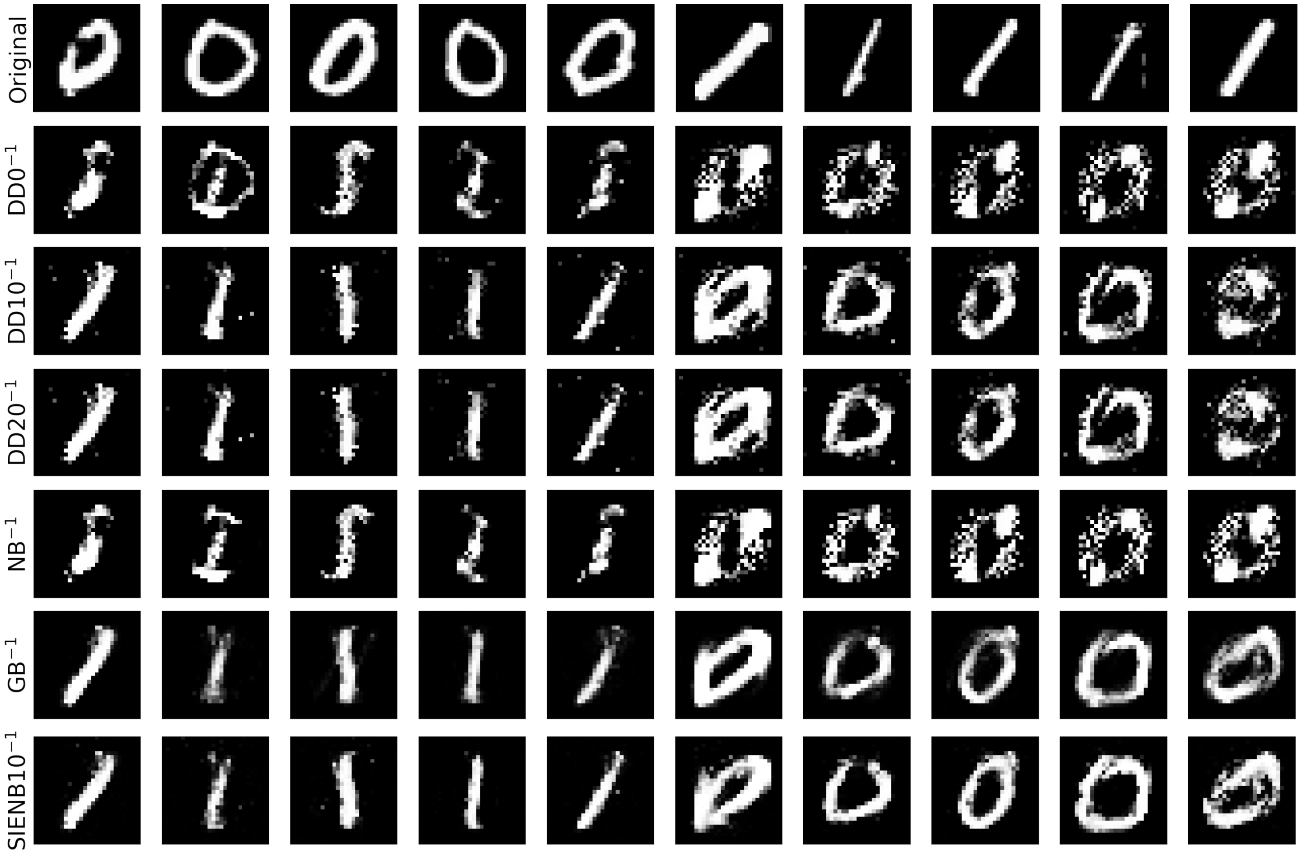

Figure 7: Original images at the top followed by flipped transformations (denoted by inverse $^{-1}$) for various algorithms, i.e., where 0s are transformed to 1s and vice versa (same labels as in Fig. 6).

F. Proofs

Theorem 1 (Observed Space Decomposition)**.** *If $T_1, T_2 \colon \mathcal{X}^m \to [0,1]^m$ are invertible transformations and we define the following auxiliary function g as: $g(\boldsymbol{x}; T_1, T_2) \triangleq \frac{|J_{T_1}(\boldsymbol{x})|}{|J_{T_1}(\boldsymbol{x})|+|J_{T_2}(\boldsymbol{x})|}$, we can decompose the mutual information in the latent space as follows:*

$$
\begin{aligned}
I(Z,Y) = & \underbrace{JSD(P_{X_1}, P_{X_2})}_{\text{Constant w.r.t. } T_1 \text{ and } T_2} \\
& + \underbrace{\tfrac{1}{2}(\mathbb{E}_{X_1}[-\log g(\boldsymbol{x}; T_1, T_2)] + \mathbb{E}_{X_2}[-\log(1 - g(\boldsymbol{x}; T_1, T_2))])}_{\text{Classification CE loss in } \textbf{observed} \text{ space}} \\
& + \underbrace{\mathrm{KL}(\tfrac{1}{2}(P_{X_1} + P_{X_2}), \tfrac{1}{2}(|J_{T_1}| + |J_{T_2}|))}_{\text{Observed mixture vs Jacobian mixture}} \\
& + \underbrace{H(\tfrac{1}{2}(P_{Z_1} + P_{Z_2}))}_{\text{Entropy of latent mixture}} - \log 2 \,.
\end{aligned} \tag{2}
$$

Proof of Theorem 1. First, we note the following:

$$
\begin{aligned}
I(Z,Y) &= \mathrm{JSD}(P_{Z_1}, P_{Z_2}) && (7)\\
&= \tfrac{1}{2}(\mathrm{KL}(P_{Z_1}, P_Z) + \mathrm{KL}(P_{Z_2}, P_Z)) \,, && (8)
\end{aligned}
$$

where $P_Z = \frac{1}{2}(P_{Z_1} + P_{Z_2})$ is the mixture distribution of P_{Z_1} and P_{Z_2} (or the distribution after marginalizing over Y)—P_X is defined similarly. The first equality is a well-known equality between mutual information and JSD. The second is just the definition of JSD. Now each of the KL terms can be expanded in terms of X_y and t_y:

$$
\begin{aligned}
& \mathrm{KL}(P_{Z_y}, P_Z) \\
& = \mathbb{E}_{Z_y}\left[\log \frac{P_{Z_y}(\boldsymbol{z})}{P_Z(\boldsymbol{z})}\right] \\
& = \mathbb{E}_{Z_y}[\log P_{Z_y}(\boldsymbol{z})] + \mathbb{E}_{Z_y}[-\log P_Z(\boldsymbol{z})] \\
& = \mathbb{E}_{X_y}[\log P_{X_y}(\boldsymbol{x}) |J_{t_y}(\boldsymbol{x})|^{-1}] + H_{\mathrm{C}}(P_{Z_y}, P_Z) \\
& = \mathbb{E}_{X_y}\left[\log \frac{P_{X_y}(\boldsymbol{x}) |J_{t_y}(\boldsymbol{x})|^{-1} P_X(\boldsymbol{x})}{P_X(\boldsymbol{x})}\right] + H_{\mathrm{C}}(P_{Z_y}, P_Z) \\
& = \mathbb{E}_{X_y}\left[\log \frac{P_{X_y}(\boldsymbol{x})}{P_X(\boldsymbol{x})}\right] + \mathbb{E}_{X_y}[\log |J_{t_y}(\boldsymbol{x})|^{-1} P_X(\boldsymbol{x})] + H_{\mathrm{C}}(P_{Z_y}, P_Z) \\
& = \mathrm{KL}(P_{X_y}, P_X) + \mathbb{E}_{X_y}[\log |J_{t_y}(\boldsymbol{x})|^{-1} P_X(\boldsymbol{x})] + H_{\mathrm{C}}(P_{Z_y}, P_Z) \,.
\end{aligned} \tag{9}
$$

We also note that the average of cross entropies is the entropy of P_Z:

$$
\begin{aligned}
& \tfrac{1}{2} H_{\mathrm{C}}(P_{Z_1}, P_Z) + \tfrac{1}{2} H_{\mathrm{C}}(P_{Z_2}, P_Z) \\
& = \int -\tfrac{1}{2} P_{Z_1}(\boldsymbol{z}) \log P_Z(\boldsymbol{z}) d\boldsymbol{z} + \int -\tfrac{1}{2} P_{Z_2}(\boldsymbol{z}) \log P_Z(\boldsymbol{z}) d\boldsymbol{z} \\
& = \int -\tfrac{1}{2}(P_{Z_1}(\boldsymbol{z}) + P_{Z_2}(\boldsymbol{z})) \log P_Z(\boldsymbol{z}) d\boldsymbol{z} \\
& = \int -P_Z(\boldsymbol{z}) \log P_Z(\boldsymbol{z}) d\boldsymbol{z} \\
& = H(P_Z) = H(\tfrac{1}{2}(P_{Z_1} + P_{Z_2})) \,.
\end{aligned} \tag{10}
$$

Now we look at the second to last term involving $\mathbb{E}_{X_1}$ and we subtract the corresponding term from the cross entropy loss to

simplify:

$$\begin{aligned}
&\mathbb{E}_{X_1}[\log|J_{t_1}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})] - \mathbb{E}_{X_1}[-\log g(\boldsymbol{x};t_1,t_2)] \\
&= \mathbb{E}_{X_1}[\log|J_{t_1}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})g(\boldsymbol{x};t_1,t_2)] \\
&= \mathbb{E}_{X_1}\left[\log|J_{t_1}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})\frac{|J_{t_1}(\boldsymbol{x})|}{|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|}\right] \\
&= \mathbb{E}_{X_1}\left[\log\frac{P_X(\boldsymbol{x})}{\frac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)}\right] + \log\tfrac{1}{2}\,.
\end{aligned} \tag{11}$$

This can be derived similarly for the terms involving $\mathbb{E}_{X_2}$:

$$\begin{aligned}
&\mathbb{E}_{X_2}[\log|J_{t_2}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})] - \mathbb{E}_{X_2}[-\log(1-g(\boldsymbol{x};t_1,t_2))] \\
&= \mathbb{E}_{X_2}\left[\log\frac{P_X(\boldsymbol{x})}{\frac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)}\right] + \log\tfrac{1}{2}\,.
\end{aligned} \tag{12}$$

By averaging the two terms from Eqn. 11 and Eqn. 11, we get a KL term:

$$\begin{aligned}
&\tfrac{1}{2}\Big(\mathbb{E}_{X_1}\left[\log\frac{P_X(\boldsymbol{x})}{\frac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)}\right] + \log\tfrac{1}{2} \\
&\qquad + \mathbb{E}_{X_2}\left[\log\frac{P_X(\boldsymbol{x})}{\frac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)}\right] + \log\tfrac{1}{2}\Big) \\
&= \int \tfrac{1}{2}P_{X_1}(\boldsymbol{x})\log\frac{P_X(\boldsymbol{x})}{\frac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)}d\boldsymbol{x} \\
&\qquad + \int \tfrac{1}{2}P_{X_2}(\boldsymbol{x})\log\frac{P_X(\boldsymbol{x})}{\frac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)}d\boldsymbol{x} - \log 2 \\
&= \int \tfrac{1}{2}(P_{X_1}(\boldsymbol{x})+P_{X_2}(\boldsymbol{x}))\log\frac{P_X(\boldsymbol{x})}{\frac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)}d\boldsymbol{x} - \log 2 \\
&= \int P_X(\boldsymbol{x})\log\frac{P_X(\boldsymbol{x})}{\frac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)}d\boldsymbol{x} - \log 2 \\
&= \mathrm{KL}(P_X, \tfrac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)) - \log 2 \\
&= \mathrm{KL}(\tfrac{1}{2}(P_{X_1}+P_{X_2}), \tfrac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)) - \log 2\,.
\end{aligned} \tag{13}$$

By combining the results above, we can derive the result from the theorem:

$$\begin{aligned}
&I(Z,Y) = \mathrm{JSD}(P_{Z_1}, P_{Z_2}) \\
&= \tfrac{1}{2}(\mathrm{KL}(P_{Z_1}, P_Z) + \mathrm{KL}(P_{Z_2}, P_Z)) \\
&= \tfrac{1}{2}\big(\mathrm{KL}(P_{X_1}, P_X) + \mathbb{E}_{X_1}[\log|J_{t_1}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})] + H_{\mathrm{C}}(P_{Z_1}, P_Z) && \text{(By Eqn. 9)} \\
&\quad + \mathrm{KL}(P_{X_2}, P_X) + \mathbb{E}_{X_2}[\log|J_{t_2}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})] + H_{\mathrm{C}}(P_{Z_2}, P_Z)\big) \\
&= \mathrm{JSD}(P_{X_1}, P_{X_2}) + \tfrac{1}{2}H_{\mathrm{C}}(P_{Z_1}, P_Z) + \tfrac{1}{2}H_{\mathrm{C}}(P_{Z_2}, P_Z) && \text{(Definition of JSD)} \\
&\quad + \tfrac{1}{2}\big(\mathbb{E}_{X_1}[\log|J_{t_1}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})] + \mathbb{E}_{X_2}[\log|J_{t_2}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})]\big) \\
&= \mathrm{JSD}(P_{X_1}, P_{X_2}) + H(\tfrac{1}{2}(P_{Z_1}+P_{Z_2})) && \text{(By Eqn. 10)} \\
&\quad + \tfrac{1}{2}\big(\mathbb{E}_{X_1}[\log|J_{t_1}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})] + \mathbb{E}_{X_2}[\log|J_{t_2}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})]\big) \\
&= \mathrm{JSD}(P_{X_1}, P_{X_2}) + \tfrac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log g(\boldsymbol{x};t_1,t_2)\right] + \mathbb{E}_{X_2}\left[-\log(1-g(\boldsymbol{x};t_1,t_2))\right]\right) \\
&\quad - \tfrac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log g(\boldsymbol{x};t_1,t_2)\right] + \mathbb{E}_{X_2}\left[-\log(1-g(\boldsymbol{x};t_1,t_2))\right]\right) \\
&\quad + \tfrac{1}{2}\big(\mathbb{E}_{X_1}[\log|J_{t_1}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})] + \mathbb{E}_{X_2}[\log|J_{t_2}(\boldsymbol{x})|^{-1}P_X(\boldsymbol{x})]\big) \\
&\quad + H(\tfrac{1}{2}(P_{Z_1}+P_{Z_2})) && \text{(Inflation by cross entropy classification term)} \\
&= \mathrm{JSD}(P_{X_1}, P_{X_2}) + \tfrac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log g(\boldsymbol{x};t_1,t_2)\right] + \mathbb{E}_{X_2}\left[-\log(1-g(\boldsymbol{x};t_1,t_2))\right]\right) \\
&\quad + \mathrm{KL}(\tfrac{1}{2}(P_{X_1}+P_{X_2}), \tfrac{1}{2}(|J_{t_1}(\boldsymbol{x})|+|J_{t_2}(\boldsymbol{x})|)) + H(\tfrac{1}{2}(P_{Z_1}+P_{Z_2})) - \log 2\,. && \text{(By Eqn. 13)}
\end{aligned}$$

□

Corollary 4 (Reduction in Classification Loss Term)**.** *The shared parts of classifier destructors (i.e., $(t_{\text{pre}}, t_{\text{post}})$) will not modify the mutual information, and the non-shared parts (i.e., $(t_{\text{mid},1}, t_{\text{mid},2})$) will reduce the classification loss term in Theorem 1, i.e. $G(t_{\text{mid},1}, t_{\text{mid},2}) \leq G(\text{id}, \text{id}) = \log 2$, where $G(t_1, t_2) = \frac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log g(\boldsymbol{x}; t_1, t_2)\right] + \mathbb{E}_{X_2}\left[-\log(1 - g(\boldsymbol{x}; t_1, t_2))\right]\right)$.*

Proof of Cor. 4. We remember the well-known invariance property of mutual information under invertible transformations: If t is an invertible function and $Z \triangleq t(X)$, then $I(X, Y) = I(Z, Y)$. Thus, both of the shared pre and post processing transformations (t_{pre} and t_{post}) do not change the mutual information.

The main idea that we want to prove next is that, if we fit a classifier f by minimizing cross entropy loss, i.e.,

$$\hat{f} = \arg\min_f \mathbb{E}_{X_1}[-\log f(\boldsymbol{x})] + \mathbb{E}_{X_2}[-\log(1 - f(\boldsymbol{x}))]\,, \tag{14}$$

then any classifier destructors $(t_{\text{mid},1}, t_{\text{mid},2})$ that are consistent with $\hat{f}$ will reduce the classification loss term from Theorem 1. Thus, we show that the middle non-shared part $(t_{\text{mid},1}, t_{\text{mid},2})$ will reduce the classification term in Theorem 1 denoted by $G(t_1, t_2)$. First let us note the following two facts:

$$g(\boldsymbol{x}; \text{id}, \text{id}) = \frac{|J_{\text{id}}|}{|J_{\text{id}}| + |J_{\text{id}}|} = \frac{1}{2} \tag{15}$$

$$\begin{aligned} G(\text{id}, \text{id}) &= \tfrac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log g(\boldsymbol{x}; \text{id}, \text{id})\right] + \mathbb{E}_{X_2}\left[-\log(1 - g(\boldsymbol{x}; \text{id}, \text{id}))\right]\right) \\ &= \tfrac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log \tfrac{1}{2}\right] + \mathbb{E}_{X_2}\left[-\log(1 - \tfrac{1}{2})\right]\right) \\ &= \log 2\,. \end{aligned} \tag{16}$$

Now we can derive the final result using the classifier consistency property and the definition of the minimization in where we assume that $f(\boldsymbol{x}) = 1/2$ is one of the possible classifers:

$$\begin{aligned} &G(t_{\text{mid},1}, t_{\text{mid},2}) \\ &= \tfrac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log g(\boldsymbol{x}; t_{\text{mid},1}, t_{\text{mid},2})\right] + \mathbb{E}_{X_2}\left[-\log(1 - g(\boldsymbol{x}; t_{\text{mid},1}, t_{\text{mid},2}))\right]\right) \\ &= \tfrac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log \hat{f}(\boldsymbol{x})\right] + \mathbb{E}_{X_2}\left[-\log(1 - \hat{f}(\boldsymbol{x}))\right]\right) && \text{(Classifier consistency Def. 1)} \\ &= \min_f \tfrac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log f(\boldsymbol{x})\right] + \mathbb{E}_{X_2}\left[-\log(1 - f(\boldsymbol{x}))\right]\right) && \text{(Classification minimization via Eqn. 14)} \\ &\leq \tfrac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log \tfrac{1}{2}\right] + \mathbb{E}_{X_2}\left[-\log(1 - \tfrac{1}{2})\right]\right) && \text{(Minimum is less than one particular value)} \\ &= \tfrac{1}{2}\left(\mathbb{E}_{X_1}\left[-\log g(\boldsymbol{x}; \text{id}, \text{id})\right] + \mathbb{E}_{X_2}\left[-\log(1 - g(\boldsymbol{x}; \text{id}, \text{id}))\right]\right) && \text{(Eqn. 15)} \\ &= G(\text{id}, \text{id}) = \log 2\,. && \text{(Eqn. 16)} \end{aligned}$$

The inequality holds as long as $f(\boldsymbol{x}) = 1/2$ is a possible function in the function class being considered—a nearly trivial constraint to satisfy. □

Proposition 2 (Naïve Bayes Classifier Destructors)**.** *Given a naïve Bayes classifier with $P_Y(1) = P_Y(2) = \frac{1}{2}$: $f(\boldsymbol{x}) = \frac{\prod_{i=1}^m \widehat{P}_{X_{1,i}}(x_i)}{\prod_{i=1}^m \widehat{P}_{X_{1,i}}(x_i) + \prod_{i=1}^m \widehat{P}_{X_{2,i}}(x_i)}$, the following feature-wise composite transformations are Wasserstein classifier destructors: $t_y(\boldsymbol{x}) = [d^{-1}_{\text{bary},1}(d_{y,1}(x_1)),\ d^{-1}_{\text{bary},2}(d_{y,2}(x_2)), \cdots, d^{-1}_{\text{bary},m}(d_{y,m}(x_m))]$, where $d_{y,i} = \text{DensDestructor}(\widehat{P}_{X_{y,i}})$, $d^{-1}_{\text{bary},i} = \text{DensDestructor}(\text{bary}(\widehat{P}_{X_{1,i}}, \widehat{P}_{X_{2,i}}))$, and $\text{bary}(\cdot, \cdot)$ denotes the Wasserstein barycenter distribution.*

Proof of Proposition 2. Let $t_{\text{pre}} = \text{id}$, $t_{\text{post}}(\boldsymbol{x}) = [d^{-1}_{\text{bary},1}(x_1),\ d^{-1}_{\text{bary},2}(x_2), \cdots, d^{-1}_{\text{bary},m}(x_m)]$, and $t_{\text{mid},y}(\boldsymbol{x}) \equiv t_y(\boldsymbol{x}) = [d_{y,1}(x_1),\ d_{y,2}(x_2), \cdots, d_{y,m}(x_m)]$ (where notation on mid is suppressed for notational simplicity) be the composite components from the definition of classifier destructors Def. 3. We first prove the classifier consistency of $(t_{\text{mid},1}, t_{\text{mid},2})$ and

the naïve Bayes classifier defined in the proposition:

$$
\begin{aligned}
g(\boldsymbol{x}; t_1, t_2) &= \frac{|J_{t_1}(\boldsymbol{x})|}{|J_{t_1}(\boldsymbol{x})| + |J_{t_2}(\boldsymbol{x})|} && \text{(Def. 3)} \\
&= \frac{\prod_{i=1}^m |J_{t_{1,i}}(x_i)|}{(\prod_{i=1}^m |J_{t_{1,i}}(x_i)| + \prod_{i=1}^m |J_{t_{2,i}}(x_i)|)} && \text{(Jacobian is diagonal because element-wise transformations)} \\
&= \frac{\prod_{i=1}^m \widehat{P}_{X_{1,i}}(x_i)}{(\prod_{i=1}^m \widehat{P}_{X_{1,i}}(x_i) + \widehat{P}_{X_{2,i}}(x_i))} && \text{(Definition of density destructors)} \\
&= f(\boldsymbol{x})\,.
\end{aligned}
$$

The Wasserstein barycenter mapping property (Def. 2) for each dimension can be established as follows. First, we know that for a univariate distribution the optimal Monge map between P_α to P_β is the composition of CDF and inverse CDF functions respectively (Peyré & Cuturi, 2019):

$$
M^*_{\alpha\to\beta} = F^{-1}_{P_\beta} \circ F_{P_\alpha}\,. \tag{17}
$$

Second, the McCann interpolation gives us the optimal transport map to the barycenter between these two distributions as (Peyré & Cuturi, 2019):

$$
\begin{aligned}
M^*_{\alpha\to\text{bary}} &= \tfrac{1}{2}(\text{id} + M^*_{\alpha\to\beta}) && (18)\\
&= \tfrac{1}{2}(\text{id} + F^{-1}_\beta \circ F_\alpha)\,. && (19)
\end{aligned}
$$

Third, the barycenter quantile function for 1D distributions is given as (Peyré & Cuturi, 2019, Remark 9.6):

$$
F^{-1}_{\text{bary}}(u) = \tfrac{1}{2}(F^{-1}_\alpha(u) + F^{-1}_\beta(u)) \tag{20}
$$

By combining these facts, we can now derive the result:

$$
\begin{aligned}
t_{\alpha,i}(x_i) &= d^{-1}_{\text{bary},i}(d_{\alpha,i}(x_i)) && \text{(By definition)} \\
&= d^{-1}_{\text{bary},i}(F_{\alpha,i}(x_i)) && \text{(CDF is 1D density destructor)} \\
&= \tfrac{1}{2}(F^{-1}_{\alpha,i}(F_{\alpha,i}(x_i)) + F^{-1}_{\beta,i}(F_{\alpha,i}(x_i))) && \text{(Eqn. 20)} \\
&= \tfrac{1}{2}(x_i + F^{-1}_{\beta,i}(F_{\alpha,i}(x_i))) \\
&= \tfrac{1}{2}(x_i + M^*_{\alpha\to\beta,i}(x_i)) && \text{(Eqn. 17)} \\
&= M^*_{\alpha\to\text{bary},i}(x_i)\,, && \text{(Eqn. 18)}
\end{aligned}
$$

where the same derivation holds for $\alpha = 1$ ($\beta = 2$) or vice versa with $\alpha = 2$ ($\beta = 1$). Thus, the full transformation has the Wasserstein barycenter mapping property.

□

Proposition 3 (Gaussian Bayes Classifier Destructors)**.** *Given a Gaussian Bayes classifier:* $f(\boldsymbol{x}) = \frac{P_{\mathcal{N}(\mu_1,\Sigma_1)}(\boldsymbol{x})}{P_{\mathcal{N}(\mu_1,\Sigma_1)}(\boldsymbol{x}) + P_{\mathcal{N}(\mu_2,\Sigma_2)}(\boldsymbol{x})}$*, the following composite transformations are Wasserstein classifier destructors:* $t_y = d^{-1}_{\text{bary}} \circ d_y$*, where* $d_y(\boldsymbol{x}) \triangleq \Phi(A_y(\boldsymbol{x} - \mu_y))$ *and* $d_{\text{bary}}(\boldsymbol{x}) \triangleq \Phi(\Sigma^{-1/2}_{\text{bary}}(\boldsymbol{x} - \mu_{\text{bary}}))$ *are density destructors for* $P_{\mathcal{N}(\mu_y,\Sigma_y)}$ *and* $P_{\mathcal{N}(\mu_{\text{bary}},\Sigma_{\text{bary}})}$ *respectively, and* $A_y \triangleq \Sigma^{-1/2}_{\text{bary}} \frac{1}{2}(I + B_y)$ *where* $B_y \triangleq \Sigma_y^{-1/2}(\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2}\Sigma_y^{-1/2}$ *is the projection matrix for the optimal Monge map between the two class distributions (if* $y = 1$*, then* $\tilde{y} = 2$ *or vice versa).*

Proof of Proposition 3. As a point of notation, we will use Φ to denote the element-wise application of the standard normal CDF function and ϕ is the element-wise density function of a standard normal function. Similarly Φ^{-1} is the inverse CDF of a standard normal function applied element-wise. First, we want to prove that the following transformations are density destructors for $P_{\mathcal{N}(\mu_y,\Sigma_y)}$ and $P_{\mathcal{N}(\mu_{\text{bary}},\Sigma_{\text{bary}})}$ respectively:

$$
\begin{aligned}
d_y(\boldsymbol{x}) &\triangleq \Phi(A_y(\boldsymbol{x} - \mu_y)) && (21)\\
d_{\text{bary}}(\boldsymbol{x}) &\triangleq \Phi(\Sigma^{-1/2}_{\text{bary}}(\boldsymbol{x} - \mu_{\text{bary}}))\,, && (22)
\end{aligned}
$$

where

$$\mu_{\text{bary}} = \tfrac{1}{2}(\mu_1 + \mu_2) \tag{23}$$
$$\Sigma_{\text{bary}} \triangleq \Sigma_1^{-1/2}(\tfrac{1}{2}(\Sigma_1 + (\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}))^2\Sigma_1^{-1/2} \tag{24}$$
$$A_y \triangleq \Sigma_{\text{bary}}^{-1/2}\tfrac{1}{2}(I + B_y) \tag{25}$$
$$B_y \triangleq \Sigma_y^{-1/2}(\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2}\Sigma_y^{-1/2}\,. \tag{26}$$

For d_y and d_{bary} to be density destructors, we must check that: $|J_{d_y}(\boldsymbol{x})| = P_{\mathcal{N}(\mu_y,\Sigma_{\tilde{y}})}$ and similarly for d_{bary}. First, we check for d_{bary}:

$$\begin{aligned}
|J_{d_{\text{bary}}}(\boldsymbol{x})| &= \phi(\Sigma_{\text{bary}}^{-1/2}(\boldsymbol{x} - \mu_{\text{bary}}))|\Sigma_{\text{bary}}^{-1/2}| \\
&= |\Sigma_{\text{bary}}^{-1/2}|(2\pi)^{-1/2}\exp(-\tfrac{1}{2}(\Sigma_{\text{bary}}^{-1/2}(\boldsymbol{x} - \mu_{\text{bary}}))^T(\Sigma_{\text{bary}}^{-1/2}(\boldsymbol{x} - \mu_{\text{bary}}))) \\
&= (2\pi|\Sigma_{\text{bary}}|)^{-1/2}\exp(-\tfrac{1}{2}(\boldsymbol{x} - \mu_{\text{bary}})^T\Sigma_{\text{bary}}^{-1}(\boldsymbol{x} - \mu_{\text{bary}})) \\
&= P_{\mathcal{N}(\boldsymbol{x};\mu_{\text{bary}},\Sigma_{\text{bary}})}\,.
\end{aligned} \tag{27}$$

This can be done similarly for d_y if we can prove that $A_{y,\tilde{y}}^T A_y = \Sigma_y^{-1}$. To do this, we will define the following matrix:

$$C_y \triangleq \tfrac{1}{2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2})\Sigma_y^{-1/2}\,. \tag{28}$$

With this definition, we can derive that

$$\begin{aligned}
&C_y^T C_y \\
&= (\tfrac{1}{2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2})\Sigma_y^{-1/2})^T(\tfrac{1}{2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2})\Sigma_y^{-1/2}) \\
&= \Sigma_y^{-1/2}\tfrac{1}{2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2})(\tfrac{1}{2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2})\Sigma_y^{-1/2}) \\
&= \Sigma_y^{-1/2}(\tfrac{1}{2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2}))^2\Sigma_y^{-1/2} \\
&= \Sigma_{\text{bary}}\,.
\end{aligned} \tag{29}$$

And we can derive our needed result:

$$\begin{aligned}
&A_y^T A_y \\
&= (\Sigma_{\text{bary}}^{-1/2}\tfrac{1}{2}\Sigma_y^{-1/2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2})\Sigma_y^{-1/2})^T \\
&\quad\times(\Sigma_{\text{bary}}^{-1/2}\tfrac{1}{2}\Sigma_y^{-1/2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2})\Sigma_y^{-1/2}) \\
&= \tfrac{1}{2}\Sigma_y^{-1/2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2}\Sigma_y^{-1/2}\Sigma_{\text{bary}}^{-1/2}\Sigma_{\text{bary}}^{-1/2} \\
&\quad\times\tfrac{1}{2}\Sigma_y^{-1/2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2})\Sigma_y^{-1/2} \\
&= \Sigma_y^{-1/2}[\tfrac{1}{2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2}\Sigma_y^{-1/2}]\Sigma_{\text{bary}}^{-1} \\
&\quad\times[\tfrac{1}{2}\Sigma_y^{-1/2}(\Sigma_y + (\Sigma_y^{1/2}\Sigma_{\tilde{y}}\Sigma_y^{1/2})^{1/2})]\Sigma_y^{-1/2} \\
&= \Sigma_y^{-1/2}C_y\Sigma_{\text{bary}}^{-1}C_y^T\Sigma_y^{-1/2} \\
&= \Sigma_y^{-1/2}C_y(C_y^TC_y)^{-1}C_y^T\Sigma_y^{-1/2} \\
&= \Sigma_y^{-1/2}C_yC_y^{-1}(C_y^T)^{-1}C_y^T\Sigma_y^{-1/2} \\
&= \Sigma_y^{-1/2}I\Sigma_y^{-1/2} \\
&= \Sigma_y^{-1}\,.
\end{aligned} \tag{30}$$

Thus, both d_y and d_{bary} are in fact density destructors for $P_{\mathcal{N}(\mu_y,\Sigma_y)}$ and $P_{\mathcal{N}(\mu_{\text{bary}},\Sigma_{\text{bary}})}$ respectively.

Given this, it is easy to show the classifier destructor property for the destructors $t_{\text{mid},y} \equiv t_y$:

$$\begin{aligned}
g(\boldsymbol{x}; t_1, t_2) &= \frac{|J_{t_1}(\boldsymbol{x})|}{|J_{t_1}(\boldsymbol{x})| + |J_{t_2}(\boldsymbol{x})|} && \text{(Definition of } g\text{)} \\
&= \frac{|J_{d_1}(\boldsymbol{x})|}{|J_{d_1}(\boldsymbol{x})| + |J_{d_2}(\boldsymbol{x})|} && \text{(Definition of transformations)} \\
&= \frac{P_{\mathcal{N}(\mu_1,\Sigma_1)}(\boldsymbol{x})}{P_{\mathcal{N}(\mu_1,\Sigma_1)}(\boldsymbol{x}) + P_{\mathcal{N}(\mu_2,\Sigma_2)}(\boldsymbol{x})} && \text{(Density destructor property)} \\
&= f(\boldsymbol{x})\,. && \text{(Definition of Gaussian Bayes classifier)}
\end{aligned}$$

Now we want to prove the Wasserstein barycenter mapping property (Def. 2, i.e., that the composition of $t_{y\to\text{bary}} = d_{\text{bary}}^{-1} \circ d_y$ is actually the barycenter optimal transformation. First we know that the optimal transport map to the barycenter is given by the middle point of McCann's interpolation (Peyré & Cuturi, 2019):

$$M^*_{y\to\text{bary}}(\boldsymbol{x}) = \tfrac{1}{2}(\boldsymbol{x} + M^*_{y\to\tilde{y}}(\boldsymbol{x}))\,, \tag{31}$$

where

$$M^*_{y\to\tilde{y}} = B_y(\boldsymbol{x} - \mu_y) + \mu_{\tilde{y}} \tag{32}$$

is the optimal transport map between y and $\tilde{y}$ (Peyré & Cuturi, 2019). Now let's start with the definition of our composite transformation and show that this composition is equivalent:

$$\begin{aligned}
t_{y\to\text{bary}}(\boldsymbol{x}) &\triangleq d_{\text{bary}}^{-1}(d_y(\boldsymbol{x})) \\
&= d_{\text{bary}}^{-1}(\Phi(A_y(\boldsymbol{x} - \mu_y))) && \text{(Eqn. 21)} \\
&= \Sigma_{\text{bary}}^{1/2}\Phi^{-1}(\Phi(A_y(\boldsymbol{x} - \mu_y))) + \mu_{\text{bary}} && \text{(Eqn. 22)} \\
&= \Sigma_{\text{bary}}^{1/2}A_y(\boldsymbol{x} - \mu_y) + \mu_{\text{bary}} && \text{(Inverse cancellation)} \\
&= \Sigma_{\text{bary}}^{1/2}\Sigma_{\text{bary}}^{-1/2}\tfrac{1}{2}(I + B_y)(\boldsymbol{x} - \mu_y) + \mu_{\text{bary}} && \text{(Eqn. 25)} \\
&= \tfrac{1}{2}(I + B_y)(\boldsymbol{x} - \mu_y) + \mu_{\text{bary}} \\
&= \tfrac{1}{2}(\boldsymbol{x} - \mu_y + B_y(\boldsymbol{x} - \mu_y)) + \mu_{\text{bary}} \\
&= \tfrac{1}{2}(\boldsymbol{x} + B_y(\boldsymbol{x} - \mu_y) + 2\mu_{\text{bary}} - \mu_y) \\
&= \tfrac{1}{2}(\boldsymbol{x} + B_y(\boldsymbol{x} - \mu_y) + (\mu_y + \mu_{\tilde{y}}) - \mu_y) && \text{(Eqn. 23)} \\
&= \tfrac{1}{2}(\boldsymbol{x} + B_y(\boldsymbol{x} - \mu_y) + \mu_{\tilde{y}}) \\
&= \tfrac{1}{2}(\boldsymbol{x} + M^*_{y\to\tilde{y}}(\boldsymbol{x})) && \text{(Eqn. 32)} \\
&\equiv M^*_{y\to\text{bary}}(\boldsymbol{x})\,. && \text{(Eqn. 31)}
\end{aligned}$$

□