Wasserstein t-SNE

Fynn Bachmann，Philipp Hennig，Dmitry Kobak

ECML-PKDD 2022

Wasserstein t-SNE

1. **SNE: Stochastic Neighbor Embedding**

2. **t-SNE**

3. **Wasserstein t-SNE**

SNE：Stochastic Neighbor Embedding - dimensionality reduction algorithm降维并且保留相似度信息

Motivation: visualizing **high-dimensional data** by giving each datapoint a location in a **two or three-dimensional map**

high-dimensional data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$

two or three-dimensional data $\mathcal{Y} = \{y_1, y_2, ..., y_n\}$

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

σ_i is the variance of the Gaussian that is centered on datapoint x_i

$$q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|y_i - y_k\|^2\right)}.$$

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

t-SNE: T-distributed stochastic neighbor embedding

$$P_{j|i} = \frac{\exp(-d_{ij}^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-d_{ik}^2/2\sigma_i^2)}$$

$$Q_{ij} = \frac{\left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2\right)^{-1}}.$$

$$P_{ij} = \frac{P_{j|i} + P_{i|j}}{2n}.$$

$$\mathrm{KL}(P\|Q) := \sum_{ij} P_{ij} \log \frac{P_{ij}}{Q_{ij}}.$$

t-SNE: T-distributed stochastic neighbor embedding

$$P_{j|i} = \frac{\exp(-d_{ij}^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-d_{ik}^2/2\sigma_i^2)}$$

$$\boxed{P_{ij} = \frac{P_{j|i} + P_{i|j}}{2n}.}$$

$$\boxed{Q_{ij} = \frac{\left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1}}{\sum_{k\neq l}\left(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2\right)^{-1}}.}$$

$$\mathrm{KL}(P\|Q) := \sum_{ij} P_{ij} \log \frac{P_{ij}}{Q_{ij}}.$$

t-distribution kernel

p_ij is symmetrized

Wasserstein metric

Definition 1 *Let (M,d) be a metric space. The p-Wasserstein distance of two distributions* μ *and* ν *is defined as*

$$W_p(\mu,\nu) := \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{M \times M} d(x,y)^p \mathrm{d}\gamma(x,y) \right)^{\frac{1}{p}}$$

where Γ *is the set of all couplings of* μ *and* ν*.*

↓

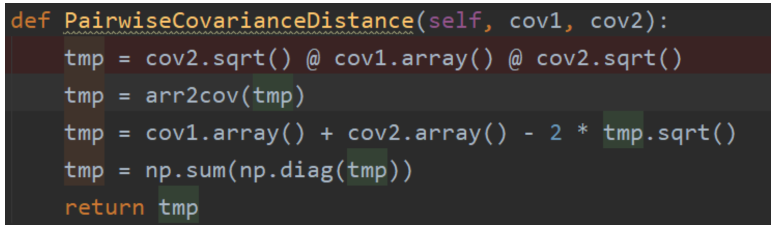
$$\begin{aligned} W_2(\mu,\nu)^2 &= \|m_1 - m_2\|_2^2 + \operatorname{tr}\left(C_1 + C_2 - 2\left(C_2^{1/2} C_1 C_2^{1/2} \right)^{1/2} \right) \\ &= \|m_1 - m_2\|_2^2 + \operatorname{tr}\left(C_1 + C_2 - 2\left(C_2 C_1 \right)^{1/2} \right). \end{aligned}$$

↓

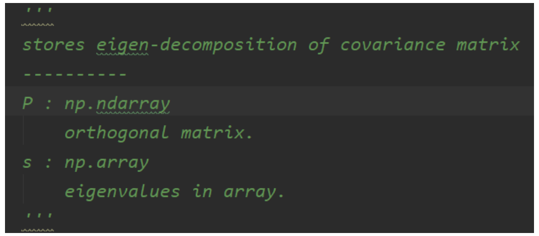
$$\tilde{W}(\mu,\nu)^2 = \boxed{(1-\lambda)} \, \|m_1 - m_2\|_2^2 + \boxed{\lambda} \, \operatorname{tr}\left(C_1 + C_2 - 2\left(C_2 C_1 \right)^{1/2} \right).$$

Computational Aspects

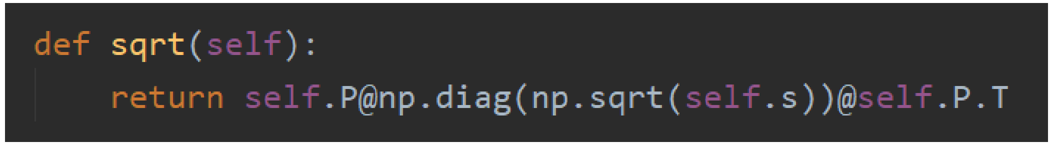
$$\begin{aligned} W_2(\mu,\nu)^2 &= \|m_1 - m_2\|_2^2 + \boxed{\operatorname{tr}\left(C_1 + C_2 - 2\left(C_2^{1/2} C_1 C_2^{1/2}\right)^{1/2}\right)} \\ &= \|m_1 - m_2\|_2^2 + \operatorname{tr}\left(C_1 + C_2 - 2\left(C_2 C_1\right)^{1/2}\right). \end{aligned}$$



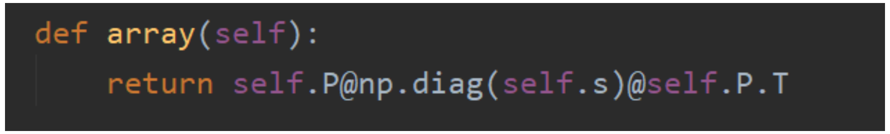
```python
def PairwiseCovarianceDistance(self, cov1, cov2):
    tmp = cov2.sqrt() @ cov1.array() @ cov2.sqrt()
    tmp = arr2cov(tmp)
    tmp = cov1.array() + cov2.array() - 2 * tmp.sqrt()
    tmp = np.sum(np.diag(tmp))
    return tmp
```



```python
'''
stores eigen-decomposition of covariance matrix
----------
P : np.ndarray
    orthogonal matrix.
s : np.array
    eigenvalues in array.
'''
```



```python
def sqrt(self):
    return self.P@np.diag(np.sqrt(self.s))@self.P.T
```



```python
def array(self):
    return self.P@np.diag(self.s)@self.P.T
```

The computational bottleneck lies in the **matrix square roots** operations. A naive method **using eigenvector decomposition** is far too time-consuming, and there is not yet, to the best of our knowledge, a straightforward way to perform it in batches on a GPU.

Boris Muzellec and Marco Cuturi. Generalizing point embeddings using the wasserstein space of elliptical distributions. Advances in Neural Information Processing Systems, 31, 2018

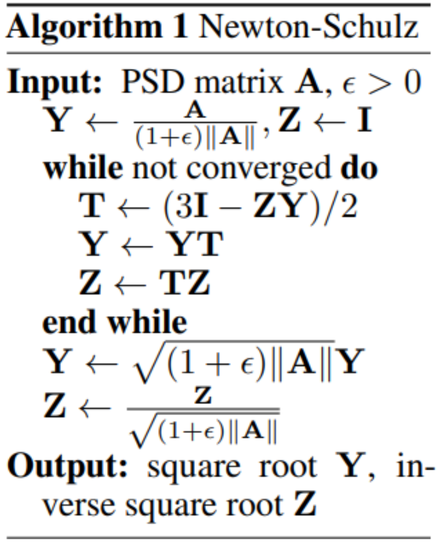
Computational Aspects

The computational bottleneck lies in the **matrix square roots** operations. A naive method **using eigenvector decomposition** is far too time-consuming, and there is not yet, to the best of our knowledge, a straightforward way to perform it in batches on **a GPU**.

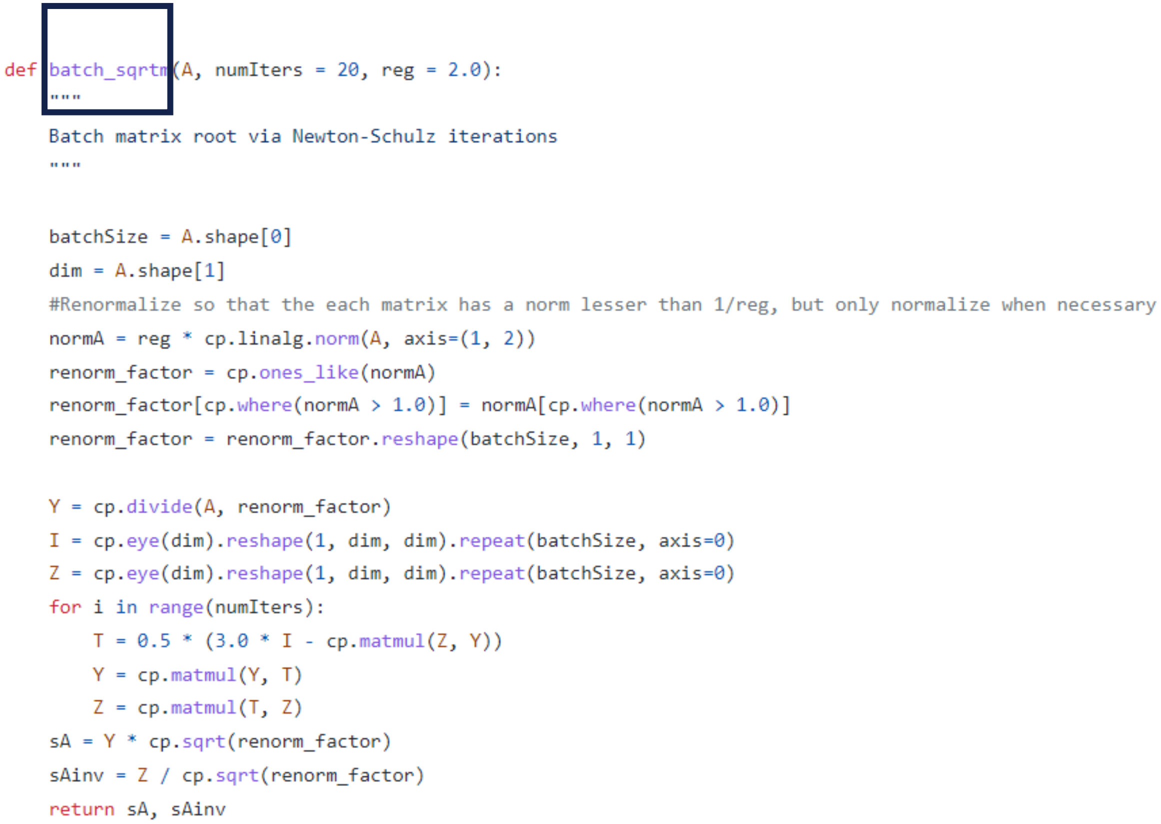
Boris Muzellec and Marco Cuturi. Generalizing point embeddings using the wasserstein space of elliptical distributions. Advances in Neural Information Processing Systems, 31, 2018

$\mathbf{A}^{\frac{1}{2}}$ $\mathbf{A}^{-\frac{1}{2}}$

↓



```
Algorithm 1 Newton-Schulz
Input: PSD matrix $\mathbf{A}$, $\epsilon > 0$
    $\mathbf{Y} \leftarrow \frac{\mathbf{A}}{(1+\epsilon)\|\mathbf{A}\|}$, $\mathbf{Z} \leftarrow \mathbf{I}$
    while not converged do
        $\mathbf{T} \leftarrow (3\mathbf{I} - \mathbf{ZY})/2$
        $\mathbf{Y} \leftarrow \mathbf{YT}$
        $\mathbf{Z} \leftarrow \mathbf{TZ}$
    end while
    $\mathbf{Y} \leftarrow \sqrt{(1+\epsilon)\|\mathbf{A}\|}\mathbf{Y}$
    $\mathbf{Z} \leftarrow \frac{\mathbf{Z}}{\sqrt{(1+\epsilon)\|\mathbf{A}\|}}$
Output: square root $\mathbf{Y}$, inverse square root $\mathbf{Z}$
```



```
def batch_sqrtm(A, numIters = 20, reg = 2.0):
    """
    Batch matrix root via Newton-Schulz iterations
    """

    batchSize = A.shape[0]
    dim = A.shape[1]
    #Renormalize so that the each matrix has a norm lesser than 1/reg, but only normalize when necessary
    normA = reg * cp.linalg.norm(A, axis=(1, 2))
    renorm_factor = cp.ones_like(normA)
    renorm_factor[cp.where(normA > 1.0)] = normA[cp.where(normA > 1.0)]
    renorm_factor = renorm_factor.reshape(batchSize, 1, 1)

    Y = cp.divide(A, renorm_factor)
    I = cp.eye(dim).reshape(1, dim, dim).repeat(batchSize, axis=0)
    Z = cp.eye(dim).reshape(1, dim, dim).repeat(batchSize, axis=0)
    for i in range(numIters):
        T = 0.5 * (3.0 * I - cp.matmul(Z, Y))
        Y = cp.matmul(Y, T)
        Z = cp.matmul(T, Z)
    sA = Y * cp.sqrt(renorm_factor)
    sAinv = Z / cp.sqrt(renorm_factor)
    return sA, sAinv
```

Wassersteint t-SNE

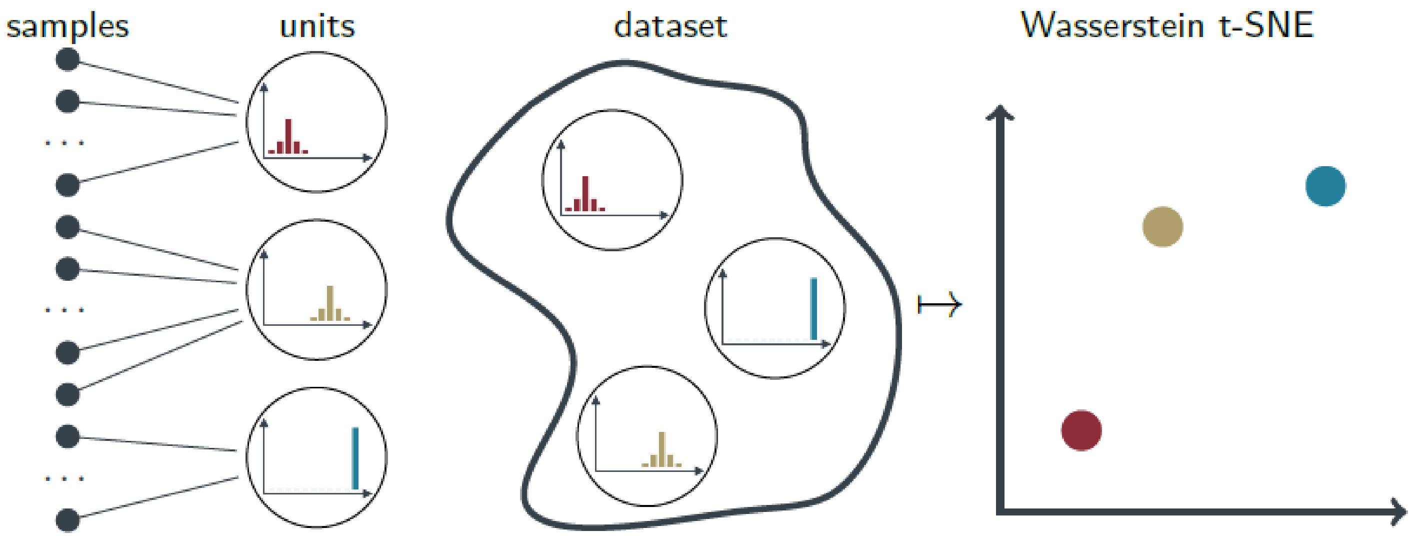
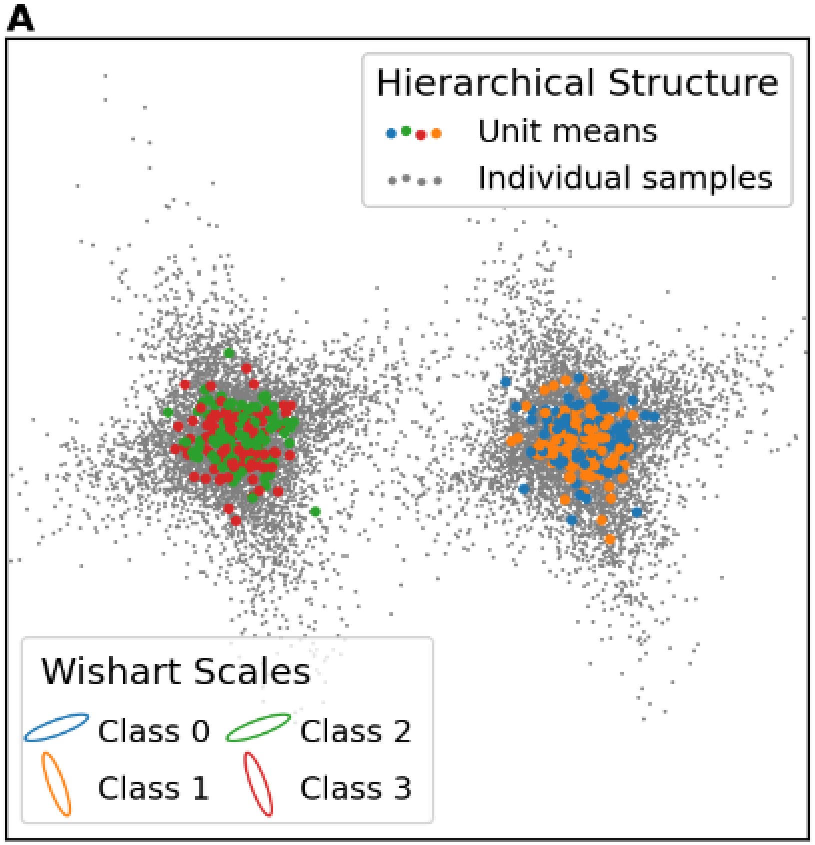
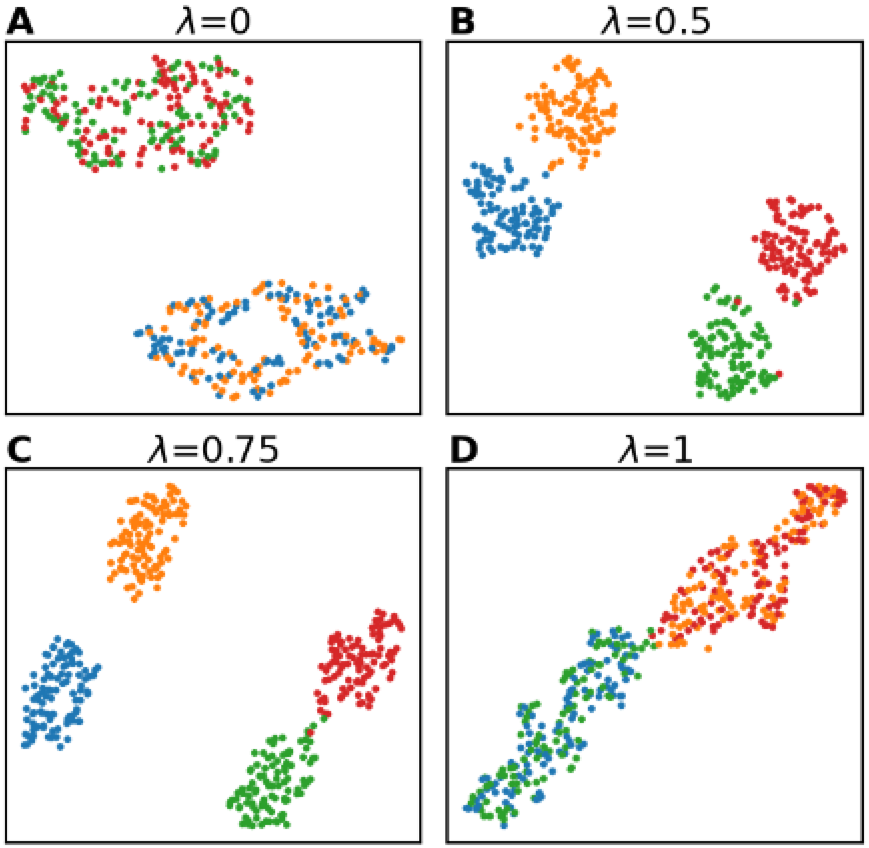


Fig. 1. Hierarchical data. Individual *samples* can be grouped into *units*. Each unit forms a probability distribution over its samples. In our *Wasserstein t-SNE* approach, units in the dataset are compared using the Wasserstein metric to construct a pairwise distance matrix, which is then embedded in two dimensions using the *t*-SNE algorithm. Units with similar probability distributions end up close together in the 2D embedding.

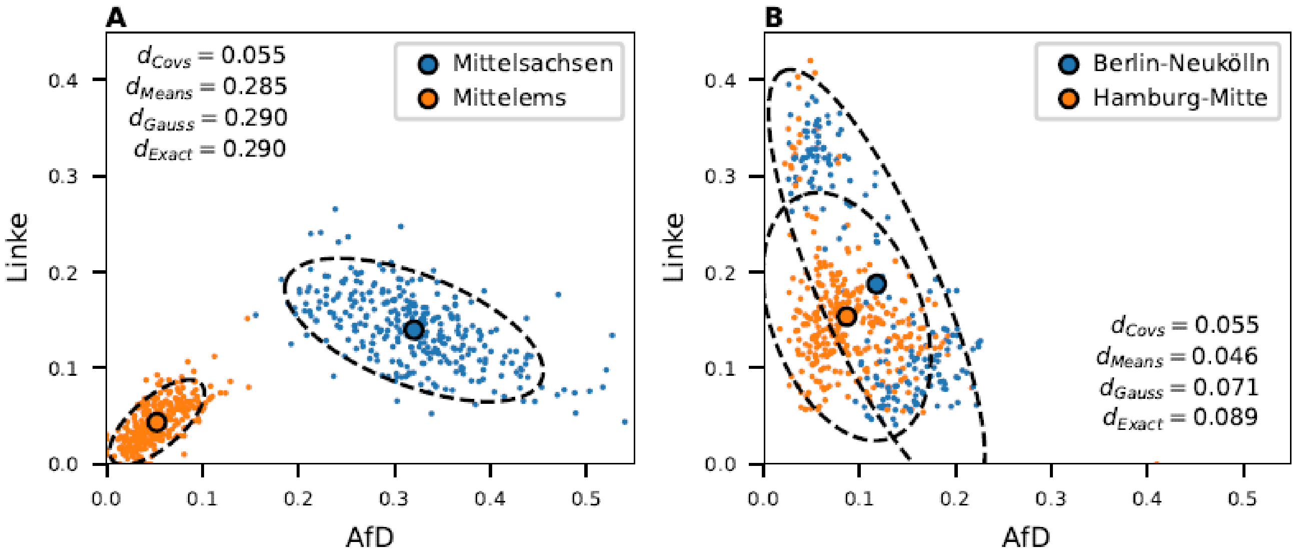
Experiments- Wasserstein t-SNE on simulated data





Experiments- German parliamentary election 2017

1. 299 voting districts (units)
2. Each unit has about 150-850 polling stations (samples).
3. The samples are represented as a points in six-dimensional space (corresponding to six political parties)



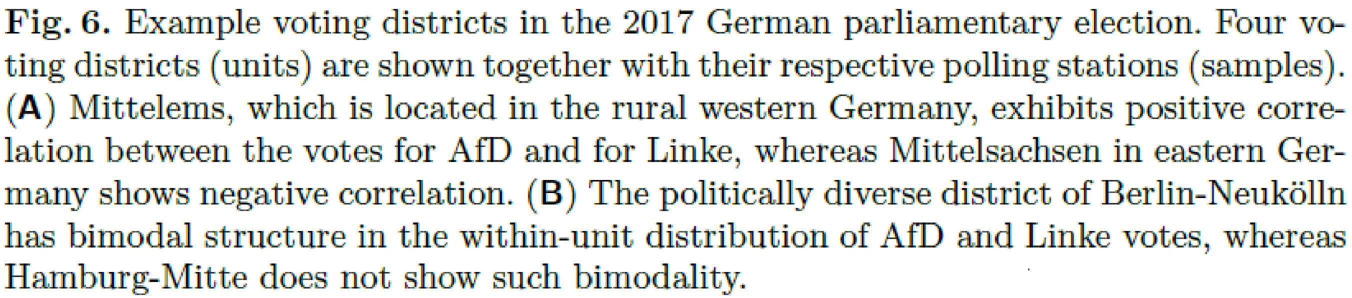


Fig. 6. Example voting districts in the 2017 German parliamentary election. Four voting districts (units) are shown together with their respective polling stations (samples). (**A**) Mittelems, which is located in the rural western Germany, exhibits positive correlation between the votes for AfD and for Linke, whereas Mittelsachsen in eastern Germany shows negative correlation. (**B**) The politically diverse district of Berlin-Neukölln has bimodal structure in the within-unit distribution of AfD and Linke votes, whereas Hamburg-Mitte does not show such bimodality.

Experiments- German parliamentary election 2017

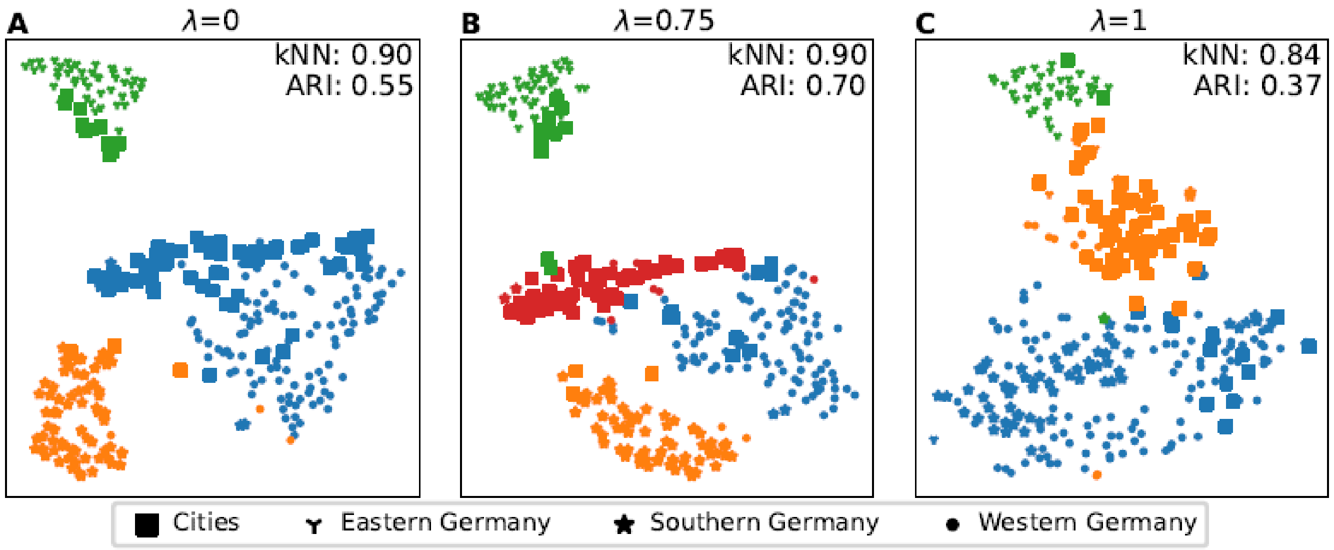
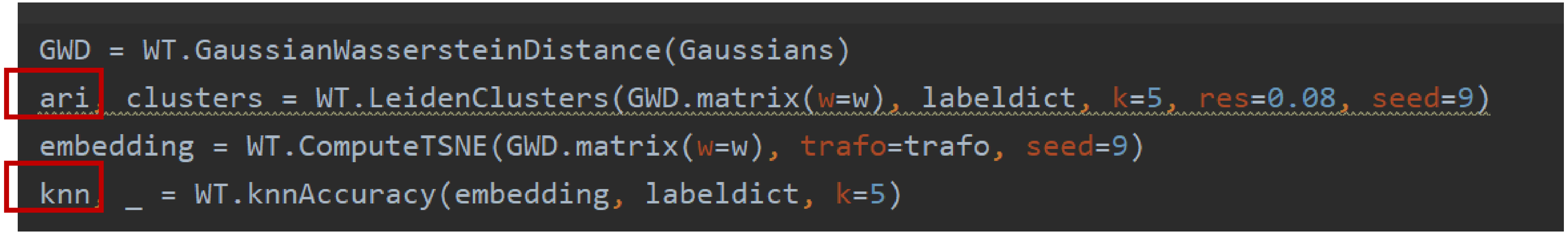


Fig. 7. Wasserstein *t*-SNE of the 2017 German parliamentary election. For clustering, we used the Leiden algorithm (on the original distance matrix) with $k = 5$ kNN graph and resolution parameter $\gamma = 0.08$. Colors correspond to the Leiden clusters; marker shape corresponds to the a priori classes. (**A**) The mean-based embedding with $\lambda = 0$



```
GWD = WT.GaussianWassersteinDistance(Gaussians)
ari, clusters = WT.LeidenClusters(GWD.matrix(w=w), labeldict, k=5, res=0.08, seed=9)
embedding = WT.ComputeTSNE(GWD.matrix(w=w), trafo=trafo, seed=9)
knn, _ = WT.knnAccuracy(embedding, labeldict, k=5)
```

Experiments- German parliamentary election 2017

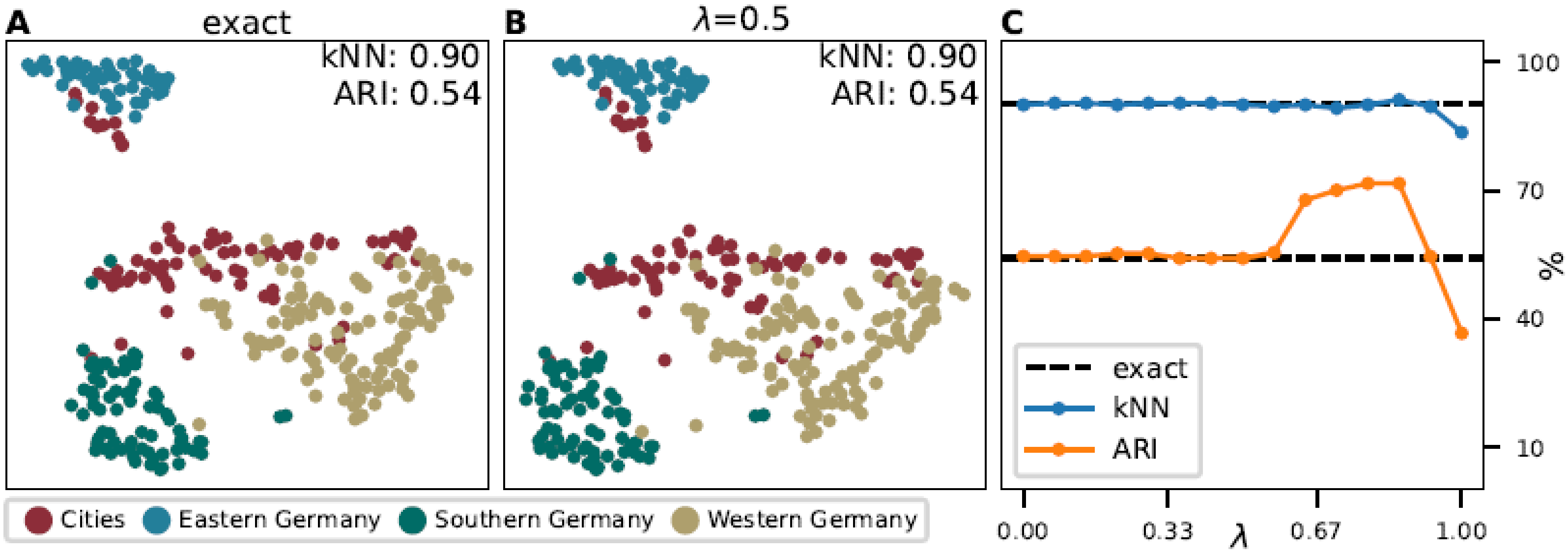


Fig. 10. Comparison of the Wasserstein *t*-SNE embeddings based on the Gaussian approximation and based on the exact Wasserstein distances. (**A**) The exact Wasserstein *t*-SNE embedding separates the classes. (**B**) The Gaussian Wasserstein embedding with $\lambda = 0.5$ shows similar structure. (**C**) The kNN classification accuracy ($k = 5$) and the ARI based on the Leiden clustering ($k = 5$ kNN graph, resolution parameter $\gamma = 0.08$) are shown for different values of λ. The black dashed lines show the kNN accuracy and the ARI of the exact Wasserstein embedding.

Computational Aspects

$$\begin{aligned} W_2(\mu,\nu)^2 &= \|m_1 - m_2\|_2^2 + \operatorname{tr}\left(C_1 + C_2 - 2\left(C_2^{1/2} C_1 C_2^{1/2}\right)^{1/2}\right) \\ &= \|m_1 - m_2\|_2^2 + \operatorname{tr}\left(C_1 + C_2 - 2\left(C_2 C_1\right)^{1/2}\right). \end{aligned}$$

If there are n units in the dataset, the $n \times n$ pairwise distance matrix can be computed in $\mathcal{O}(n^2 d^3)$ time.

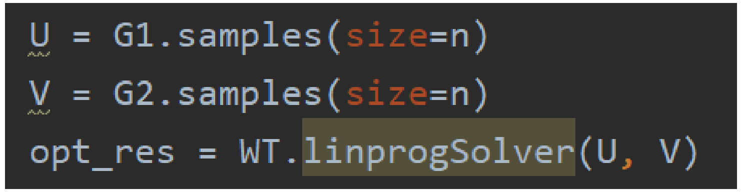
Computational Aspects

注：这里debug一下看看sample的怎么算的 设定为几，看看是跟d有关系还是m有关系

$$W_p(\mu,\nu)^p := \min_{\gamma\in\Gamma(\mu,\nu)} \sum_{M\times M} d(m_i,m_j)^p \cdot \gamma(m_i,m_j),$$

which is equivalent to the following linear program [16]:

$$\begin{array}{rrcl|rrcl} & \textbf{primal form:} & & & & \textbf{dual form:} & & \\ \text{minimize} & z & = & \mathbf{c}^T\mathbf{x}, & \text{maximize} & \tilde{z} & = & \mathbf{b}^T\mathbf{y}, \\ \text{so that} & \mathbf{A}\mathbf{x} & = & \mathbf{b} & \text{so that} & \mathbf{A}^T\mathbf{y} & \leq & \mathbf{c}. \\ \text{and} & \mathbf{x} & \geq & \mathbf{0} & & & & \end{array}$$



```
U = G1.samples(size=n)
V = G2.samples(size=n)
opt_res = WT.linprogSolver(U, V)
```

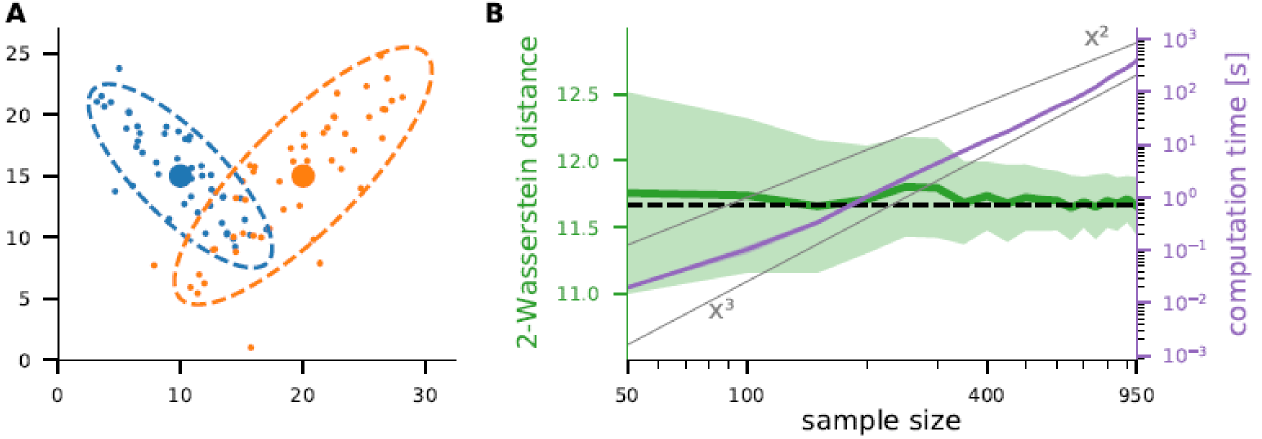
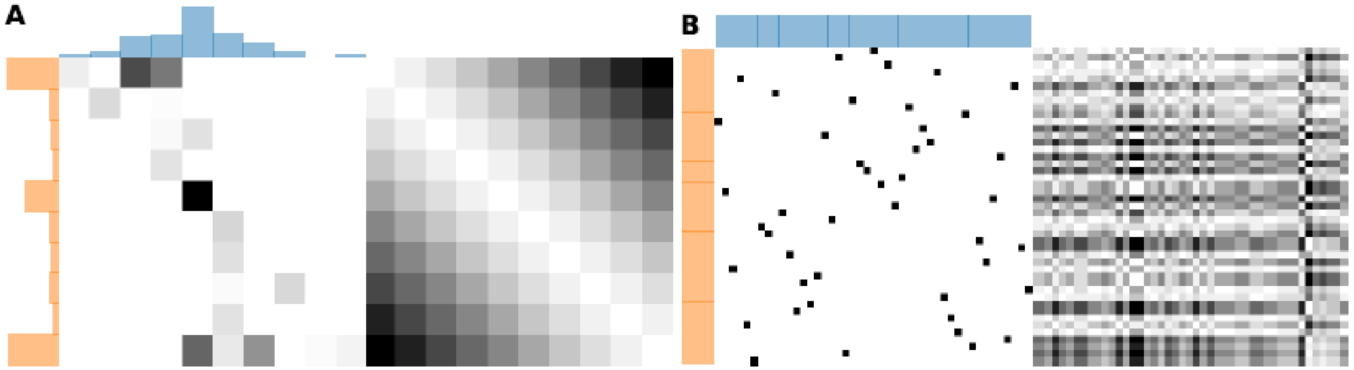


Fig. 3. Computation time and accuracy. (**A**) Two multivariate Gaussian distributions with 50 samples each. (**B**) The Wasserstein distance between the two probability distributions is computed using a different number of samples. The ground-truth distance is obtained by the closed-form solution and is shown with the dashed black line. The Wasserstein distance estimates using our linear program approach are shown in green (mean and standard deviation over 50 repetitions). The purple line shows the average runtime.

the runtime increases approximately as $\mathcal{O}(m^3)$

Computational Aspects

支撑Support指的是使概率分布有正的密度的随机变量的取值的集合



pairwise L_p-distance between all samples

Fig. 2. Wasserstein distance as a linear program. (**A**) The optimal transport map γ of two probability distributions ν (orange) and μ (blue) is shown. The heatmap represents the cost matrix C. (**B**) The same distributions can be visualized as a collection of samples, which have different support. The distance between samples ν_i and μ_j is given in the cost matrix entry C_{ij}. The size of the optimization variable γ is then upper bounded by the product of the sample sizes.

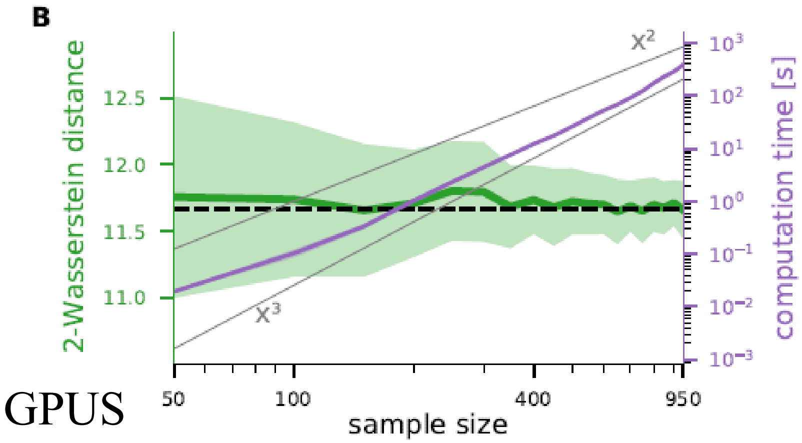
The number of variables in the transport map therefore grows exponentially with the number of features, thus this approach is intractable for datasets with many features.

we consider both distributions uniformly distributed over their samples.

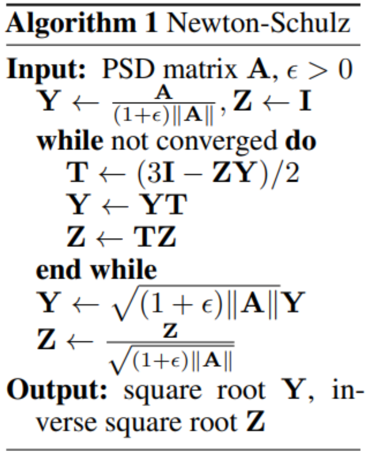
The method is only applicable when the number of samples is not large

Future work/Summary

1.We developed an approach **that scales with the number of samples** and allows to compute Wasserstein distances between samples from continuous distributions. The feature dimensionality of the Gaussians does not play a role here.



2. Backpropagation is limited on GPUS



Algorithm 1 Newton-Schulz

```
Input: PSD matrix $\mathbf{A}$, $\epsilon > 0$
  $\mathbf{Y} \leftarrow \frac{\mathbf{A}}{(1+\epsilon)\|\mathbf{A}\|}$, $\mathbf{Z} \leftarrow \mathbf{I}$
  while not converged do
    $\mathbf{T} \leftarrow (3\mathbf{I} - \mathbf{ZY})/2$
    $\mathbf{Y} \leftarrow \mathbf{YT}$
    $\mathbf{Z} \leftarrow \mathbf{TZ}$
  end while
  $\mathbf{Y} \leftarrow \sqrt{(1+\epsilon)\|\mathbf{A}\|}\mathbf{Y}$
  $\mathbf{Z} \leftarrow \frac{\mathbf{Z}}{\sqrt{(1+\epsilon)\|\mathbf{A}\|}}$
Output: square root $\mathbf{Y}$, inverse square root $\mathbf{Z}$
```