Learning Without Labels (1)
Fundamentals of Unsupervised Learning

An-Che Liang

Data Science Study Group, November 2025

Acknowledgments

Portions of this material are adapted from the following course:

- Stanford CME 250: Introduction to Machine Learning (Winter 2019)

Unsupervised Learning

Unsupervised learning focuses on discovering patterns and structure from unlabeled data. Common approaches include:

1. **Clustering**: Grouping samples based on similarity.
2. **Dimensionality Reduction**: Learning compact, low-dimensional representations of high-dimensional data.
3. **Generative Modeling**: Producing new samples that resemble those in the original dataset (to be discussed later).

Unsupervised learning is especially useful when datasets contain only features or input variables, but no corresponding labels.

Unsupervised Learning Applications

Document Clustering with LDA

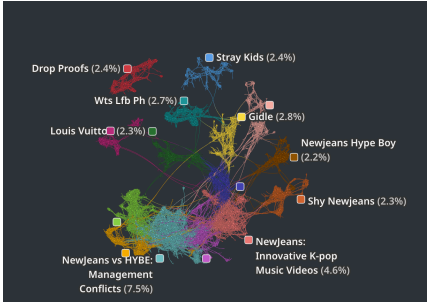
Latent Dirichlet Allocation (LDA) is a generative probabilistic model used to uncover hidden "topics" in large collections of text documents. In this unsupervised setting, we don't know:

1. What the topics are
2. Which documents belong to which topics—there are no predefined correct answers.

Because no ground truth is provided, LDA relies on unsupervised learning techniques to identify meaningful patterns directly from the documents themselves.

Unsupervised Learning Applications Cont.

Below is an LDA topic network generated using Quid Discover, showing clusters of social media posts about the K-pop group NewJeans. Note that each topic is *discovered*, rather than pre-defined.



K-means Clustering

One of the most widely used clustering algorithms is K-means, which partitions samples into K clusters by minimizing the within-cluster variation.
Suppose the within-cluster variation of cluster C_k is denoted by $W(C_k)$. Our goal is to find a set of clusters $C = \{C_1, C_2, \ldots, C_K\}$ that minimizes the total within-cluster variation:

$$\underset{C}{\operatorname{argmin}} \sum_{k=1}^{K} W(C_k).$$

K-means Clustering (Cont.)

The within-cluster variation can be written as:

$$W(C_k) = \sum_{x_i \in C_k} \|x_i - \mu_k\|_2^2,$$

where μ_k denotes the mean (centroid) of cluster C_k. Using this definition, the K-means objective becomes:

$$\underset{C}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|_2^2.$$

Lloyd's algorithm

We typically solve K-means using an iterative algorithm (often called *Lloyd's algorithm*):

1. **Initialization**: Choose K initial cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$ (e.g., randomly select K data points).
2. **Assignment step**: For each data point x_i, assign it to the closest centroid:

$$C_k = \{x_i : \|x_i - \mu_k\|_2^2 \leq \|x_i - \mu_j\|_2^2 \ \forall j \in \{1, \ldots, K\}\}.$$

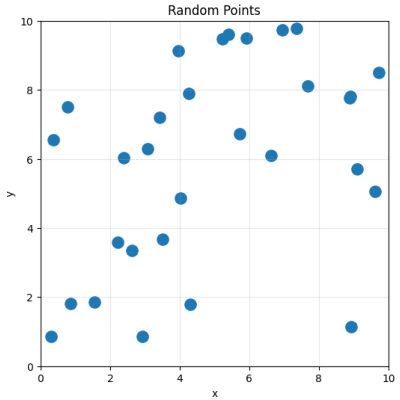
3. **Update step**: Recompute each centroid as the mean of all points assigned to that cluster:

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i, \quad k = 1, \ldots, K.$$

4. **Repeat**: Alternate between the assignment step and the update step until convergence.

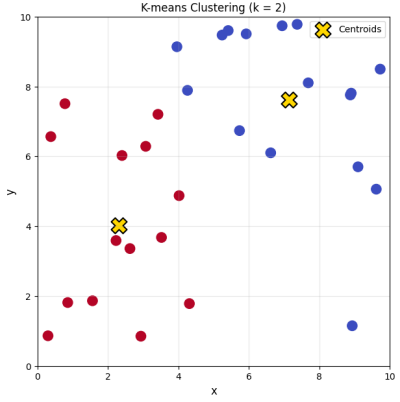
Data Preparation

We can generate random points using NumPy via the function `np.random.uniform(0, v, n)`, which samples n values uniformly from the range $[0,v]$ These points can then be visualized on a scatter plot, as shown below:



Applying K-means Clustering

We can apply K-means clustering by creating a `sklearn.cluster.KMeans(n_clusters=2)` model and fitting it to our dataset. The resulting visualization shows the two clusters along with their corresponding centroids:



Discussion on K-means

Here are some important questions to consider:

1. How do we choose the number of clusters K?
2. Is the algorithm stable? (i.e., does the same input always produce the same clustering result?)
3. How do we evaluate the quality of the clustering results?

t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a dimensionality reduction technique designed for visualizing high-dimensional data. It was developed by Laurens van der Maaten in 2008, a former student of Turing Award laureate Geoffrey Hinton and currently a researcher at Anthropic. For details, refer to the original paper, *Visualizing Data using t-SNE* published in JMLR (with over 62k citations).

t-SNE (Cont.)

For each point x_i, t-SNE defines a conditional probability that x_j is a neighbor of x_i:

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

After computing all $p_{j|i}$, the joint probabilities are symmetrized:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}.$$

t-SNE (Cont.)

How is σ_i chosen?
t-SNE sets a separate σ_i for each point by matching a user-defined *perplexity* value. The perplexity of the distribution $P_{j|i}$ is defined as:

$$\mathrm{Perp}(P_{j|i}) = 2^{H(P_{j|i})} \quad H(P_{j|i}) = -\sum_j p_{j|i} \log_2 p_{j|i}.$$

For each x_i, t-SNE performs a binary search on σ_i such that:

$$\mathrm{Perp}(P_{j|i}) = \text{desired perplexity}.$$

This ensures that each point has a similar effective number of neighbors.

t-SNE (Cont.)

Given low-dimensional embeddings $(y_1, \ldots, y_N)$, similarities are computed using a Student t-distribution with one degree of freedom:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}.$$

t-SNE (Cont.)

t-SNE finds low-dimensional embeddings $\{y_i\}$ by minimizing the KL divergence between the high-dimensional and low-dimensional similarity distributions:

$$\mathcal{L} = \mathrm{KL}(P \,\|\, Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

This objective can be optimized using gradient descent. The gradient with respect to a point y_i is:

$$\frac{\partial \mathcal{L}}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}.$$

The embedding is then updated iteratively, with η as the learning rate.

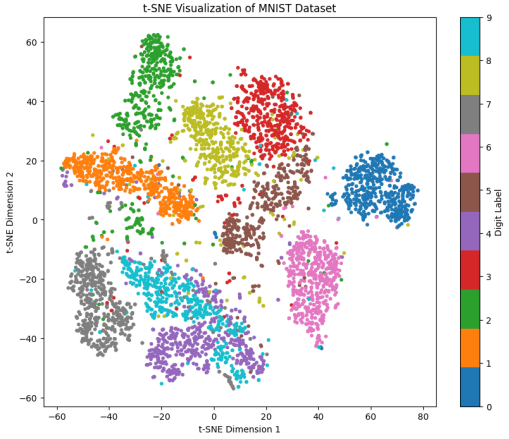
$$y_i^{(t+1)} = y_i^{(t)} - \eta \, \frac{\partial \mathcal{L}}{\partial y_i^{(t)}},$$

Preparing the MNIST Dataset

We use the MNIST dataset to illustrate the performance of t-SNE. MNIST consists of grayscale digit images with a high-dimensional input space ($28 \times 28 = 784$ features). Use `torchvision.datasets.MNIST` to load the dataset, and apply `torchvision.transforms` to normalize the pixel values before applying t-SNE.

Applying t-SNE

We can apply t-SNE using `sklearn.manifold.TSNE`. The visualization below shows the 2D embedding, with points colored according to their labels:



Discussion on t-SNE

Here are some important questions to consider:

1. How does t-SNE compare to traditional methods such as PCA?
2. What is the computational complexity of t-SNE?
3. How can we evaluate the quality of the dimensionality reduction results?

Remarks

Unsupervised learning is a powerful technique, especially when ground-truth labels are unavailable. In traditional machine learning, unsupervised methods are commonly applied to clustering and dimensionality reduction.
In modern deep learning, however, **self-supervised learning** has emerged as a major approach for leveraging large amounts of unlabeled data, enabling models to learn representations and generate text or images without explicit annotations.