Unsupervised Learning

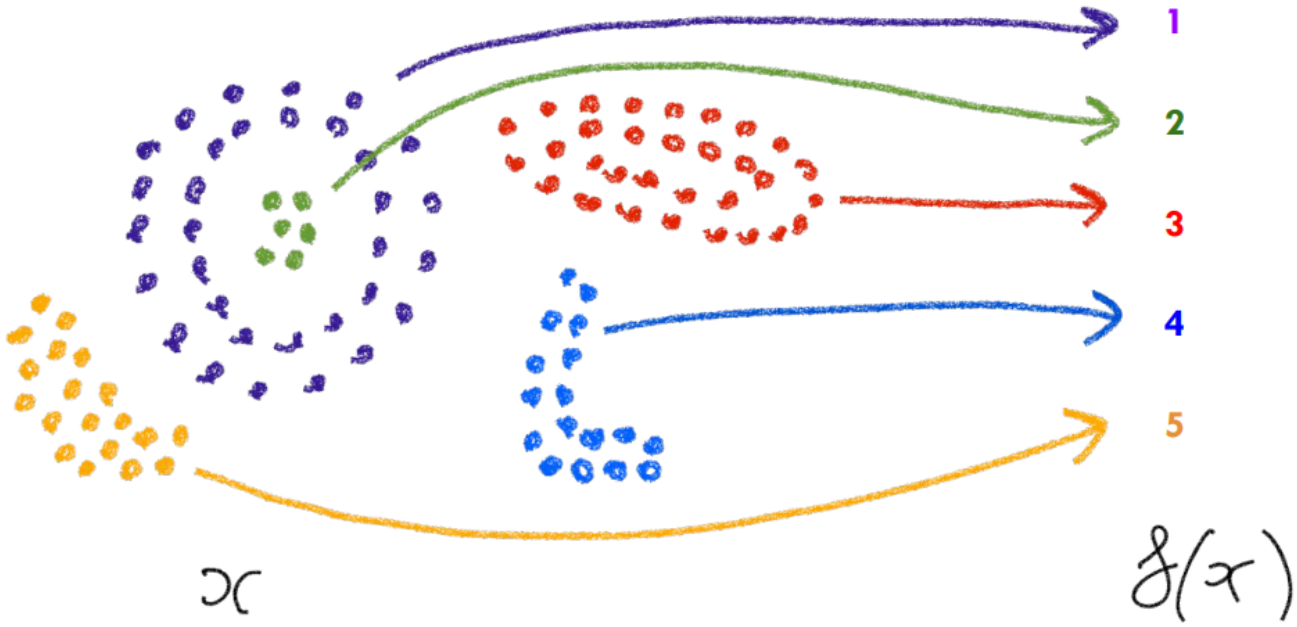
Definition

Building a model using examples without labels that learns to predict new examples

Given $T = x_1, x_m$ [1] output a hidden structure behind the various x values, that being the #clusters .

#Clustering

Find a function $f \in \mathbb{N}^X$ that assigns each input $x \in X$ a cluster index $f(x) \in \mathbb{N}$. All the points mapped to the same index form a cluster.



Why?

Clustering allows us to analyse data by grouping together data points that exhibit some regular pattern or similarity under some predefined criterion, and it can be used to compress data by reducing the number of data points as opposed to reducing the feature dimensionality.

There are several situations in which it could be applied, such as:

- Clustering users by preference[2]
- Grouping gene families from gene sequences
- Finding communities on social networks
- Image segmentation[3]

- Anomaly detection
  - Analyse a set of events or objects and flag some of them as being unusual or atypical[4]

K-Means Clustering

Given data points $X = [x_1, \ldots, x_n] \in \mathbb{R}^{d \times n}$, pix a number of clusters k and find a partition of data points into k sets minimising the variation within each set: $\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} V(C_j)$
The variation $V(C_j)$ is typically given by $\sum_{i \in C_j} ||x_i - \mu_j||^2$ and the centroid μ_j is computed with $\frac{1}{|C_j|} \sum_{i \in C_j} x_i$.

Properties

> **Note**
>
> Note that K-means clustering is sensitive to the scale of the features. Feature normalisation is a necessity in the case of features with different scales.

The K-means clustering algorithm is guaranteed to converge, as it strictly improves the objective if there is at least one cluster change and the set of possible partitions is finite.

> **Does each step of k-means move towards reducing the loss function (or at least not increasing it)?**
>
> Idea:
>
> 1. Assignment - Any other assignment would lead to a larger loss
> 2. Centroid computation - The mean of a set of values minimises the squared error.

It is not, on the other hand, guaranteed to find the global minimum; only a local one.

> **Does this mean that k-means will always find the minimum loss/clustering?**
>
> No. It will find *a* minimum. Unfortunately, the k-means loss function is generally not convex and for most problems has many minima. We are only guaranteed to find one of them.

Initial Centroid Selection

Results can vary drastically based on random seed selection; some seeds can result in a poor convergence rate or in convergence to suboptimal clusterings. Some common heuristics include:

- Random points[5] in the space
- Randomly pick examples
- Points least similar to any existing centre[6]
- Try out multiple starting points
- Initialise with the results of another clustering method

Running Time

For step 1, that is the assignment step, the time complexity is $O(kn)$. For step 2, the centroid computation, the time complexity is $O(n)$.

#Dimensionality-Reduction

Find a function $f \in Y^X$ mapping each (high-dimensional) input $x \in X$ to a lower dimensional embedding $f(x) \in Y$, where $dim(Y) \ll dim(X)$.

Why?

Using #dimensionality-reduction we can compress the data by reducing the feature dimensionality while still preserving as much data as possible[7]. This reduces the time needed for subsequent data elaboration and/or storage and allows for better visualisation of the data as well as reducing the curse of dimensionality.

Density Estimation

Find a probability distribution $f \in \Delta(X)$ that fits the data $x \in X$.

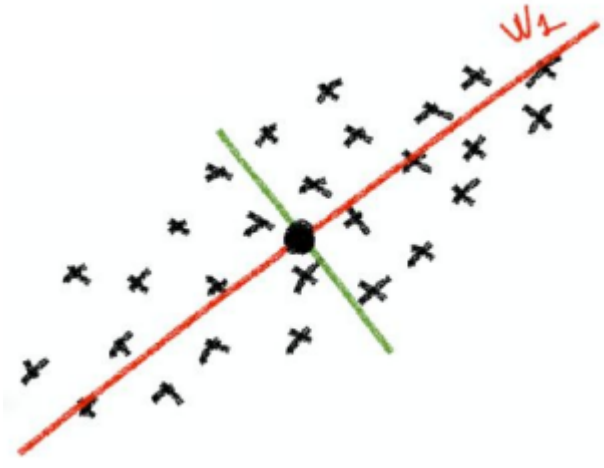
Why?

Density estimation is a method used to get an explicit estimate of the unknown probability distribution that generated the training data, thus enabling not only the generation of new data by sampling from the estimated distribution but also the detection of anomalies/novelties in terms of data points that exhibit low probabilities according to the estimated distribution.

Principal Component Analysis[8]

Calculating the i-th Principal Component

Given $w_i \in argmax\{w^T C w : w^T w = 1, w \perp w_j for\ 1 \leq j < i\}$, where $w^T C w$ shows the variance along w in the below graph, it can be shown that i-th largest eigenvalue of C is the variance along the i-th principal component, which itself is the corresponding eigenvector.



> **Alternative Interpretation**
>
> The first principal component can be interpreted as the line in space with minimal squared distance from the data points.

Dimensionality Reduction Using PCA

Let $\hat{W} = [w_1, \dots, w_k]$ hold the first k principal components derived from data points $\bar{X} = [\bar{x}_1, \dots, \bar{x}_n]$. We change this to a reduced coordinate system with the k principal components as axes. We have $T = \hat{W}^T \bar{X} \in \mathbb{R}^{k \times n}$ where T is the set of principal component scores.

How Many Principal Components?

The number of components for dimensionality reduction depends on the goal and application. There are no ways of validating it unless we use it in the context of a supervised method[9]. Despite this we can compute the cumulative proportion of explained variance, which for the first principal components is given by $\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{m} C_{jj}}$ in the case of eigenvalue decomposition, and by $\frac{\sum_{j=1}^{k} s_j^2}{\sum_{ij} \bar{X}_{ji}^2}$ in the case of SVD[10]. This all allows us to estimate the amount of information loss.

Deep Generative Models

Generative vs Discriminative Models

A #generative-model is a statistical model of the data distribution p_X or p_{XY}[11] whereas a #discriminative-model is a statistical model of the conditional distribution $p_{Y|X}$ of the target given the input. A #discriminative-model can be constructed from a #generative-model via the Bayes rule, though the inverse is not true. $p_{Y|X}(y|x) = \frac{p_{xy}(x,y)}{\sum_{y'} p_{XY}(x,y')}$

Density Estimation

Explicitly

Find a probability estimation $f \in \Delta(Z)$ that fits the data, where $z \in Z$ is sampled from an unknown data distribution $p_{data} \in \Delta(Z)$. In #supervised-learning $Z = X \times Y$, but in #unsupervised-learning $Z = X$

Implicitly

Find a function $f \in Z^{\Omega}$ that generates data $f(\omega) \in Z$ from an input ω sampled from some predefined distribution $p_{\omega} \in \Delta(\Omega)$ in a way that the distribution of generated samplse fits the (unknown) data distribution $p_{data} \in \Delta(Z)$

Objective

Define a hypothesis space $H \subset \Delta(Z)$ of models that can represent probability distributions (be it implicitly or explicitly). Define a divergence measure $d \in \mathbb{R}^{\Delta(Z) \times \Delta(Z)}$ between probability distributions in $\Delta(Z)$[12]. Find a hypothesis $q^* \in H$ that fits the data distributed according to p_{data}. The best fit is measured using the divergence d, i.e, $q^* \in argmin_{q \in H} d(p_{data}, q)$

Variational Autoencoders

#Autoencoders are a way of compressing high-dimensional data into a lower-dimensional representation by compressing the data given in input such that meaningful factors of variations in the data are preserved.

Encoders can be trained using decoders mapping the representation back to the input domain yielding $\hat{x}$ (reconstruction), so the encoder "autoencodes" its input[13]. The objective is to minimise the divergence between the input x and the reconstruction $\hat{x}$. Once we have finished training we no longer need the encoder as its only use was to help us estimate the encoder. The encoder can now be used to initialise or precompute features for supervised models.

> Is an #autoencoder a #generative-model ?
>
> The decoder could be used to generate new data, but it will not generate data according to the data distribution p_{data} when given random inputs ω

Given parameters θ, an original distribution p_{ω}, and a decoder $q_{\theta}(x|\omega)$, we can form the equation $q_{\theta} = \mathbb{E}_{\omega \sim p^{\omega}}[q_{\theta}(x|\omega)]$ and set our objective as $\theta^* \in argmin_{\theta \in \Theta} d(q_{\theta}, p_{data})$ using KL divergence, $d_{KL}(p, q) = \mathbb{E}_{x \sim p}[log \frac{p(x)}{q(x)}]$

Conditional Variational Autoencoders

Assuming we have side information $y \in Y$[14] and we want to generate new data conditioned on the side information[15], we can modify the encoder and decoder to take the side information in input obtaining $q_\psi(\omega|x,y)$ and $q_\theta(x|\omega,y)$ and then define priors conditioned on the side information $p_\omega(\omega|y)$

Issues With Variational Autoencoders

Underfitting

In the initial stages, the regulariser is too strong and tends to annihilate the model's capacity to learn.

Blurry Samples

The generator tends to produce blurry data.

Generative Adversarial Networks[16]

#GANs enable the possibility of estimating implicit densities. Assuming we have a prior density $p_\omega \in \Delta(\Omega)$ and a generator/decoder $g_\theta \in X^\Omega$ that generates data points in X given a random element from θ, the density induced by the prior p_ω and the generator g_θ is given by $q_\theta(x) = \mathbb{E}_{\omega \sim p^\omega}\delta[g_\theta(\omega) - x]$, where δ is the Dirac delta function.

The (original) #GAN objective is to find θ^* such that q_{θ^*} best fits the data distribution p_{data} under the Jenson-Shannon divergence: $\theta^* = \in argmin_\theta d_{JS}(p_{data}, q_\theta)$ where $d_{JS}(p,q) = \frac{1}{2}d_{KL}(p, \frac{p+q}{2}) + \frac{1}{2}d_{KL}(q, \frac{p+q}{2})$.

If we let $t_\phi(x)$ be a classifier[17] for data points in the training set, we get the following lower bound on our objective: $d_{JS}(p_{data}, q_\theta) = log(2) + \frac{1}{2}max_t\{\mathbb{E}_{x\sim p_{data}}[log\, t(x)] + \mathbb{E}_{x\sim q_\theta}[log(1-t(x))]\}$ $\geq log(2) + \frac{1}{2}max_\phi\{\mathbb{E}_{x\sim p_{data}}[log\, t_\phi(x)] + \mathbb{E}_{x\sim q_\theta}[log(1-t\phi(x))]\}$ which is minimised to obtain the generator's parameters: $\theta^* \in argmin_\theta\, max_\phi\{\mathbb{E}_{x\sim p_{data}}[log\, t_\phi(x)] + \mathbb{E}_{x\sim q_\theta}[log(1-t_\phi(x))]\}$ Unfortunately, this still depends on the explicit density. Luckily we can rewrite it using our generator g: $\theta^* \in argmin_\theta\, max_\phi\{\mathbb{E}_{x\sim p_{data}}[log\, t_\phi(x)] + \mathbb{E}_{\omega\sim p_\omega}[log(1-t_\phi(g_\theta(\omega)))]\}$

#GAN Objective

The objective of a #GAN can be seen as a two-player, zero-sum, non-cooperative game where player 1[18] tries to generate data that cannot be distinguished from the true data, and player 2[19] tries to guess whether the input comes from the true distribution or from player 1. This can be solved with gradient descent and specialised approaches for handling the min-max problem.

Issues with #GANs

Training Stability

Parameters might oscillate and never converge.

Mode Collapse

The generator might learn to perfectly generate a few examples from the training set.

Vanishing Gradient

If the discriminator is very successful it leaves the generator with little gradient to learn from.

Several #GANs

More GAN-like models can be constructed by considering different divergences between probabilities:

- #GANs build on Bergmann divergences.
- Wasserstein #GANs use the Wasserstein metric.

#GANs and #VAEs [20] can be combined (VAE-GAN).

Conditional #GANs exist that work akin to continual VAEs.

---

1. no y because there are no labels ↩
2. e.g., based on movie ratings ↩
3. distinguishing different parts or layers of an image ↩
4. credit card fraud detection, video surveillance ↩
5. not examples ↩
6. furthest centres heuristic ↩
7. the way information loss is measured yields different algorithms ↩
8. PCA ↩
9. e.g., reducing the input dimensionality for a supervised algorithm ↩
10. Singular Value Decomposition ↩
11. depending on the availability of target data ↩
12. Kullback-Leibler divergence ↩
13. We would have an encoder taking input x and producing output ω, and then the decoder that takes ω as input and generates $\hat{x}$, an approximation of x ↩
14. digit labels, attributes, etc ↩
15. generate the digit 7, or generate a face with glasses ↩
16. GANs ↩
17. or discriminator ↩
18. the generator ↩
19. the discriminator ↩
20. Variational AutoEncoders ↩