

10707

Deep Learning: Spring 2020

Andrej Risteski

Machine Learning Department

Lecture 8:

Intro to unsupervised
learning

Unsupervised learning

Learning from data **without** labels.

What can we hope to do:

Task A: Fit a parametrized **structure** (e.g. clustering, low-dimensional subspace, manifold) to data to reveal something meaningful about data. (**Structure learning**)

Task B: Learn a (parametrized) **distribution** *close* to data generating distribution. (**Distribution learning**)

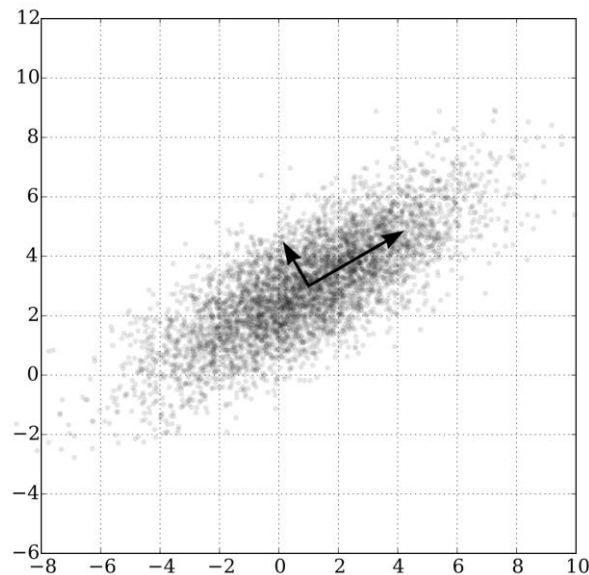
Task C: Learn a (parametrized) distribution that implicitly reveals an **“embedding”/“representation”** of data for downstream tasks. (**Representation/feature learning**)

Entangled! The “structure” and “distribution” often reveals an embedding.

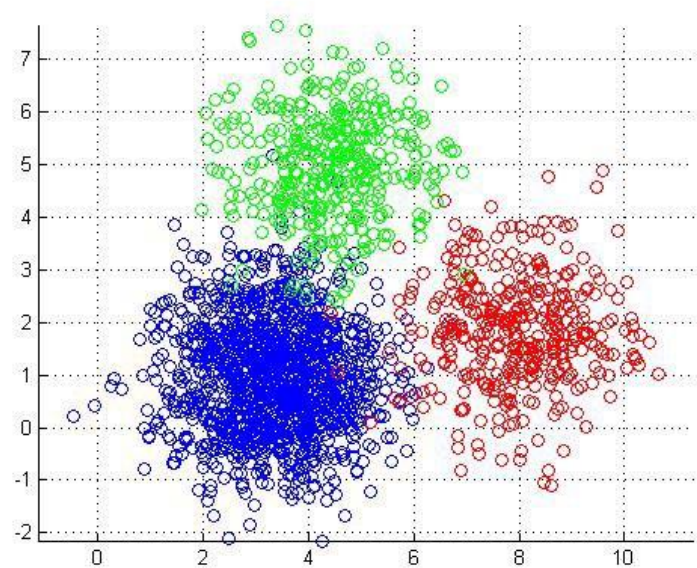
Structure learning

Fit a parametrized **structure** (e.g. clustering, low-dimensional subspace) to data to reveal something meaningful about data.

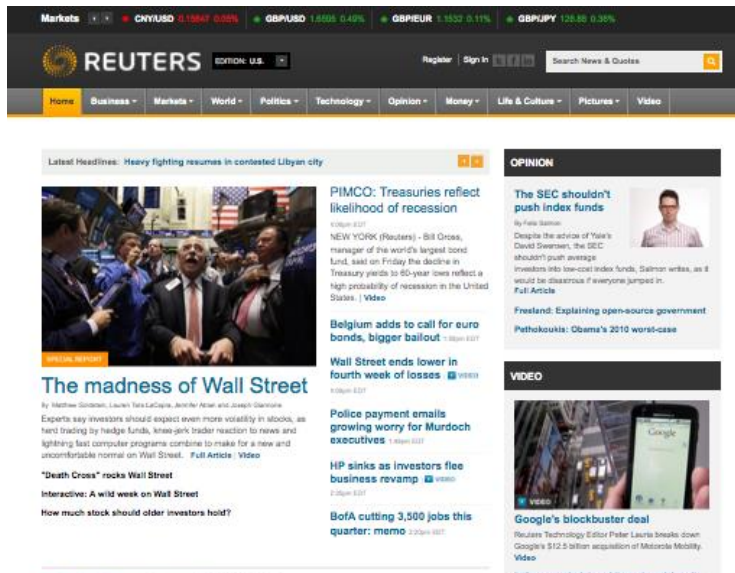
PCA(principal component analysis),
direction of highest variance



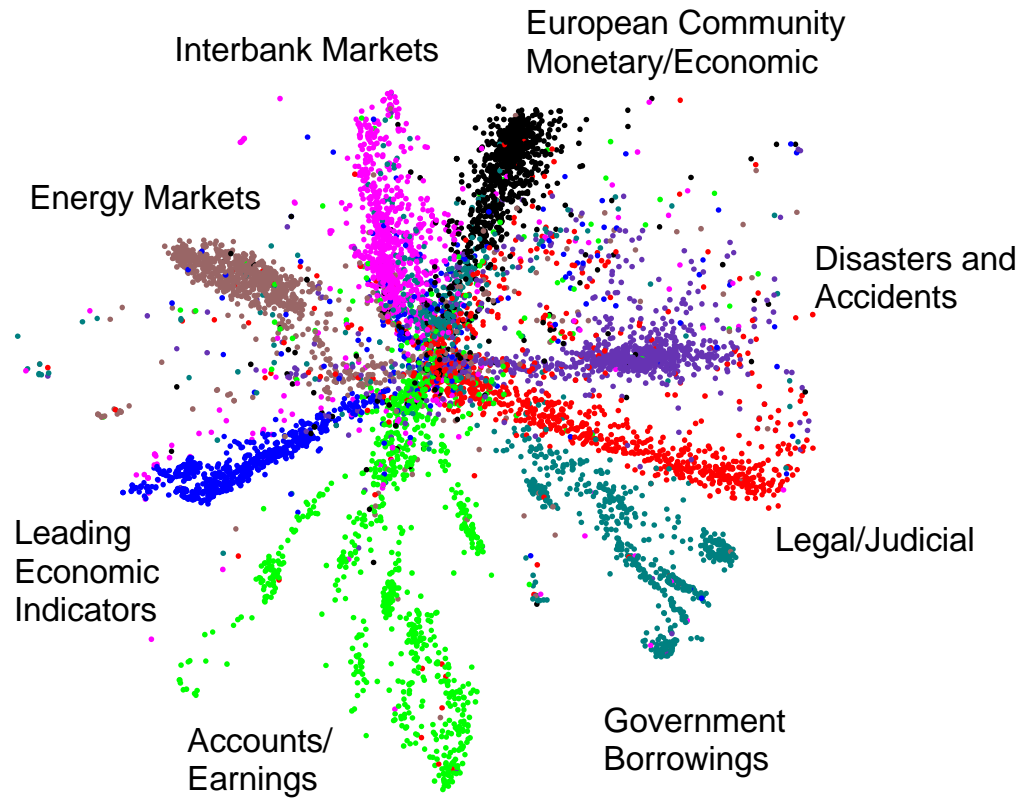
Clustering



Structure learning



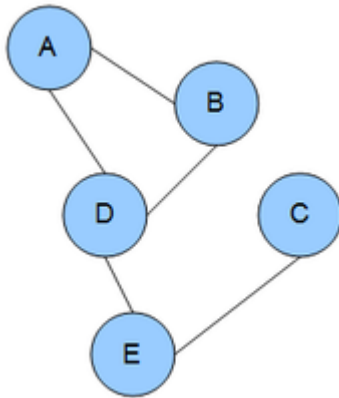
804,414 newswire stories



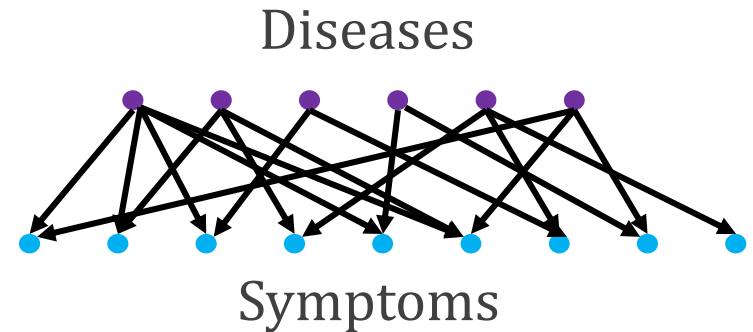
Distribution learning

Some typical choices of parametrized distributions:

Classical choices: **fully-observed** graphical models (undirected and directed), **latent-variable** graphical models (mixture models, sparse coding, topic models).



Markov Random Fields:
sparse independence
structure: “A is independent of
other vars, given B, D”

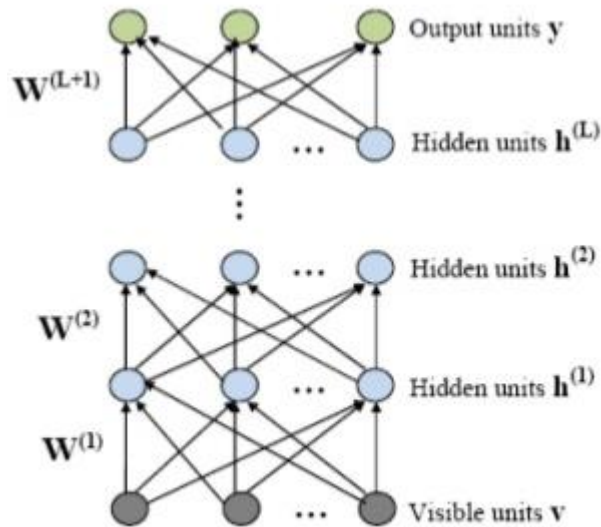


Latent variable models: data is
“simple” conditioned on some
unobserved (latent) variables

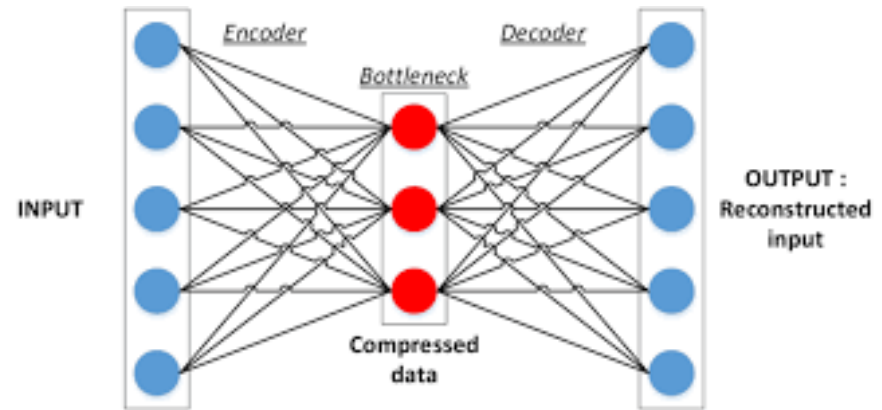
Distribution learning

Some typical choices of parametrized distributions:

Semi-modern choices: deep Boltzmann machines, deep belief networks, (variational) auto-encoders, energy models.



Deep Boltzmann machines, belief networks: graphical model analogues of deep neural networks.

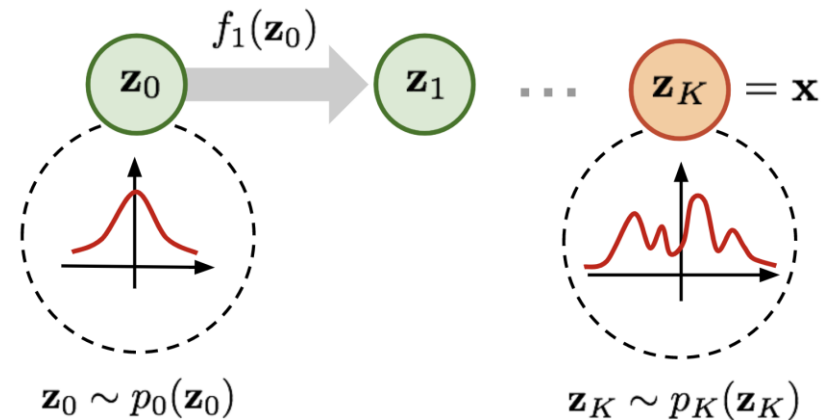
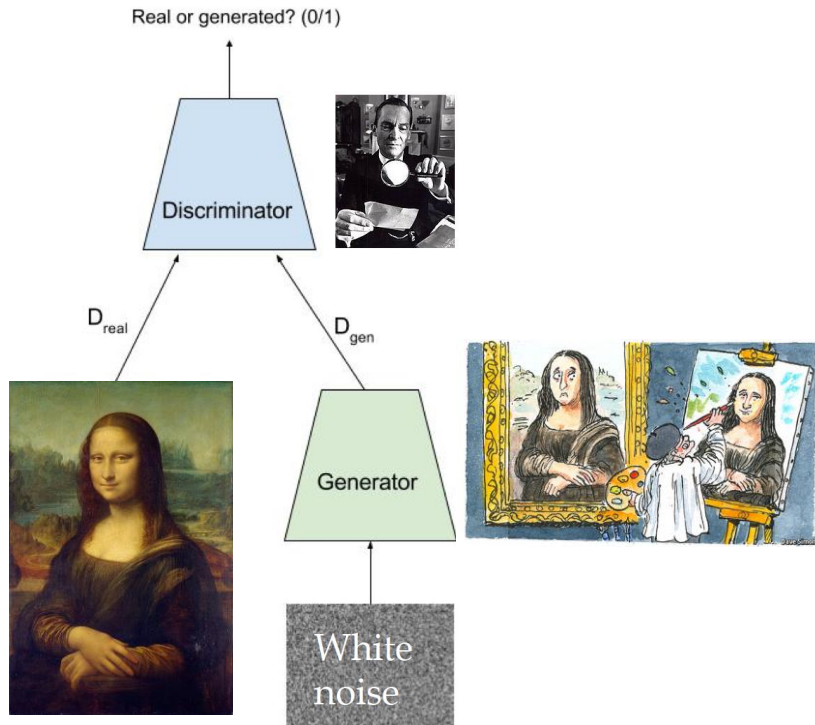


(Variational) autoencoders: model data by enforcing a latent space “bottleneck”:

Distribution learning

Some typical choices of parametrized distributions:

Modern choices: generative adversarial networks, autoregressive models (pixelRNN, pixelCNN), flow models, etc.



Distribution learning



Training
Data(CelebA)



Model Samples (Karras et.al.,
2018)

4 years of progression on Faces



2014



2015



2016



2017

Brundage et al.,
2017

Distribution learning



Whichfaceisreal.com

Distribution learning



BigGAN, Brock et al '18

Distribution learning

Conditional generative model $P(\text{zebra images} | \text{horse images})$



Style Transfer



Input Image



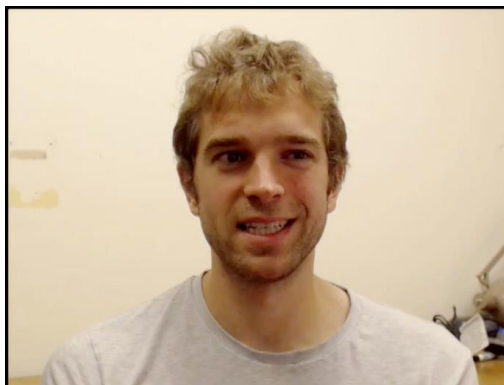
Monet



Van Gogh

Distribution learning

Source
actor



Target
actor



Real-time Reenactment



Real-time
reenactment

Reenactment Result

Representation learning and self-supervised learning

Given **unlabeled** data, **design supervised tasks** that induce a good representation for downstream tasks.

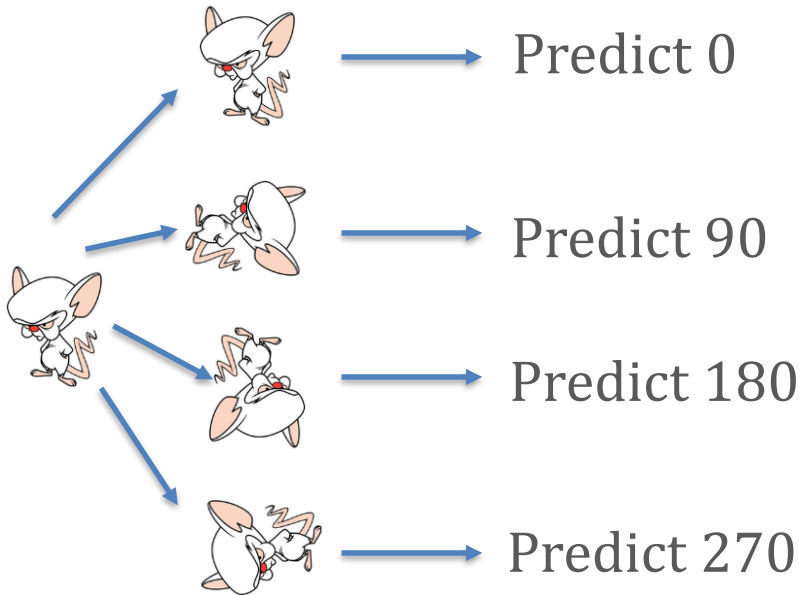
No good mathematical formalization, but the intuition is to “force” the predictor used in the task to learn something “**semantically meaningful**” about the data.

Examples in NLP: predict next word, given previous 5 words; predict middle word, given surrounding 5 words; etc.

Representation learning and self-supervised learning

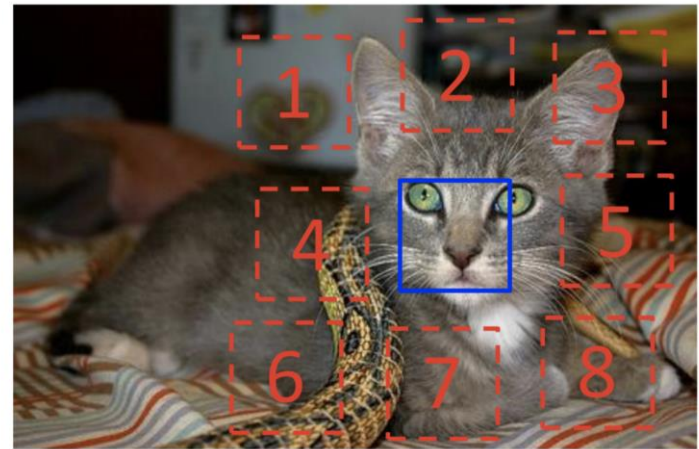
Examples in vision: a lot, and quite different in nature.

Rotation prediction



Predict one of four angles
an image is rotated by

Jigsaw puzzles



$$X = \left(\begin{array}{c} \text{cat face} \\ \text{cat ear} \end{array} \right); Y = 3$$

Predict position of second
piece wrt to first

Desiderata for a good representation

Semantically meaningful: if we train a linear classifier on top of these features, should work well for (many, most?) reasonable tasks.

“Disentangled”: in generative setting, can “vary” features independently, and generate an output that is reasonable.

(Contraversial and not well understood: are the features disentangled in priors, posteriors? Models can be reparametrized to produce arbitrarily bad entangling:

Locatello et al, Best paper award at ICML’19)

Improve sample complexity: if we wish to use feature to train a supervised model on top of, good representation could save us on sample complexity.

(See e.g. Arora-Risteski <https://arxiv.org/pdf/1706.04601.pdf>)

Relationships between the tasks

Structure learning and feature learning often blend

E.g. low dimensional features in PCA or cluster a point belongs to in clustering can be viewed as features.

Structure learning/feature learning is in general weaker than distribution learning:

E.g. methods like PCA/clustering can't be used to generate new samples.

Feature, not a bug: it has been argued that self-supervised learning works because the task we are solving is **easier** (both computationally and statistically)

Relationships between the tasks

Distribution learning often implies representation learning:

Many distributions we fit are **latent-variable** models (i.e. model the joint distribution between some latent variables \mathbf{h} and the observed data \mathbf{x})

$$P_{\theta}(x, h) = P_{\theta}(h)P_{\theta}(x|h)$$


The latent variables are often viewed as a “**representation**”.

The **posterior** distribution $P_{\theta}(h|x)$ captures a distribution over representations, given some values of the observed data.

However, a-priori, distribution $P_{\theta}(h|x)$ is a-priori not an easy distribution to **approximate/sample** from!

$$P_{\theta}(h|x) = \frac{P_{\theta}(x|h)p(h)}{\int_{h'} p(h')p(x|h')}$$

*Hard high-dimensional
sum/integral*



Representation learning and distribution learning

Common ways to get a handle of $P_\theta(h|x)$: **variational methods** and **MCMC** methods.

Question: how accurately do we need to approximate $P_\theta(h|x)$ if we want a good representation?

(Arora-Risteski blogpost: <http://www.offconvex.org/2017/06/26/unsupervised1/>)

Answer: Let $Q_\theta(h|x)$ be an approximation of $P_\theta(h|x)$ s.t.

$$KL(Q_\theta(h|x) || P_\theta(h|x)) \leq \epsilon$$

(KL is a natural metric, as a lot of the **variational methods** operate in this metric. Stay tuned!)

(Recall the definition of KL divergence: $KL(q || p) = \mathbb{E}_q \log \left(\frac{q}{p} \right)$)

Representation learning and distribution learning

Common ways to get a handle of $P_\theta(h|x)$: variational methods and MCMC methods.

Question: how accurately do we need to approximate $P_\theta(h|x)$ if we want a good representation?

Answer: Let $Q_\theta(h|x)$ be an approximation of $P_\theta(h|x)$ s.t.

$$KL(Q_\theta(h|x) || P_\theta(h|x)) \leq \epsilon$$

Suppose the way we will use the representation h is to solve **classification tasks**. Namely, let's say the labels for task are:

$$(x_t, c(h_t)), \text{ where } (h_t, x_t) \sim P_\theta(h, x) \text{ and } c(h_t) \in \{\pm 1\}$$

*In other words, the labels are a function of the **latent** variables, and the observables are the data.*

Representation learning and distribution learning

Let $Q_\theta(h|x)$ be an approximation of $P_\theta(h|x)$ s.t.

$$KL(Q_\theta(h|x) || P_\theta(h|x)) \leq \epsilon$$

Suppose the way we will use the representation h is to solve classification tasks. Namely, let's say the binary classification task is:

$$(x_t, c(h_t)), \text{ where } (h_t, x_t) \sim P_\theta(h, x) \text{ and } c(h_t) = \{\pm 1\}$$

The natural way to measure the distance with an eye towards classification tasks is total variation distance:

$$TV(Q_\theta(\cdot | x) || P_\theta(\cdot | x)) = \sup_{\Omega} \left| \Pr_{h \sim Q_\theta(\cdot | x)}[\Omega] - \Pr_{h \sim P_\theta(\cdot | x)}[\Omega] \right|$$

Why is this useful? Take $\Omega = \{\text{"C(h) is the correct label"}\}$

$\Pr_{h \sim Q_\theta(\cdot | x)}[\Omega]$ is the probability of having the correct answer using $Q_\theta(\cdot | x)$

$\left| \Pr_{h \sim Q_\theta(\cdot | x)}[\Omega] - \Pr_{h \sim P_\theta(\cdot | x)}[\Omega] \right|$ is the *difference in probability* of having the correct answer between $Q_\theta(\cdot | x)$!

Representation learning and distribution learning

Let $Q_\theta(h|x)$ be an approximation of $P_\theta(h|x)$ s.t.

$$KL(Q_\theta(h|x) || P_\theta(h|x)) \leq \epsilon$$

Suppose the way we will use the representation h is to solve classification tasks. Namely, let's say the binary classification task is:

$$(x_t, c(h_t)), \text{ where } (h_t, x_t) \sim P_\theta(h, x) \text{ and } c(h_t) = \{\pm 1\}$$

The natural way to measure the distance with an eye towards classification tasks is total variation distance:

$$TV(Q_\theta(\cdot | x) || P_\theta(\cdot | x)) = \sup_{\Omega} \left| \Pr_{h \sim Q_\theta(\cdot | x)}[\Omega] - \Pr_{h \sim P_\theta(\cdot | x)}[\Omega] \right|$$

Why is this useful? Take $\Omega = \{C(h) \text{ is the correct label}\}$

Hence, $TV(Q_\theta(\cdot | x) || P_\theta(\cdot | x)) \leq \epsilon'$ implies that the difference in performance b/w using $P_\theta(\cdot | x)$ and $Q_\theta(\cdot | x)$ is at most ϵ' (on the input sample x)

Hence, $\mathbb{E}_x[TV(Q_\theta(\cdot | x) || P_\theta(\cdot | x))] \leq \epsilon'$ implies that the difference in performance on the classification task is at most ϵ' .

Representation learning and distribution learning

Let $Q_\theta(h|x)$ be an approximation of $P_\theta(h|x)$ s.t.

$$KL(Q_\theta(h|x) || P_\theta(h|x)) \leq \epsilon$$

Suppose the way we will use the representation h is to solve classification tasks. Namely, let's say the binary classification task is:

$$(x_t, c(h_t)), \text{ where } (h_t, x_t) \sim P_\theta(h, x) \text{ and } c(h_t) = \{\pm 1\}$$

The natural way to measure the distance with an eye towards classification tasks is total variation distance:

$$TV(Q_\theta(\cdot | x) || P_\theta(\cdot | x)) = \sup_{\Omega} \left| \Pr_{h \sim Q_\theta(\cdot | x)}[\Omega] - \Pr_{h \sim P_\theta(\cdot | x)}[\Omega] \right|$$

To conclude, by Pinsker's inequality, we have

$$TV(Q_\theta(\cdot | x) || P_\theta(\cdot | x)) \leq \sqrt{\frac{1}{2} KL(Q_\theta(\cdot | x) || P_\theta(\cdot | x))} \leq \sqrt{\frac{1}{2} \epsilon}$$

So, if we want a good performance on classification task, we need an extremely close approximation to the posterior!

Training techniques: Likelihood-based vs likelihood-free

Two typical families of training algorithms:

Likelihood-based: maximize the likelihood of the data under the model (possibly with some approximations)

$$\max_{\theta} \sum_{\text{samples } x_i} \log p_{\theta}(x_i)$$

(Some statistical, some algorithmic motivations)

Training techniques: Likelihood-based vs likelihood-free

Two typical families of training algorithms:

Likelihood-based: maximize the likelihood of the data under the model (possibly with some approximations)

Typical approximations used: variational inference (optimize tractable deterministic approximation of posteriors), MCMC methods (idea: approximate difficult quantities like posteriors with sampling)

Likelihood-free: use a surrogate loss – e.g. in GANs, train a discriminator to tell real and generated samples apart; noise-contrastive training: encourage model to put probability mass away from “fake” samples.

The classics: PCA

Figure 1: Population structure within Europe.

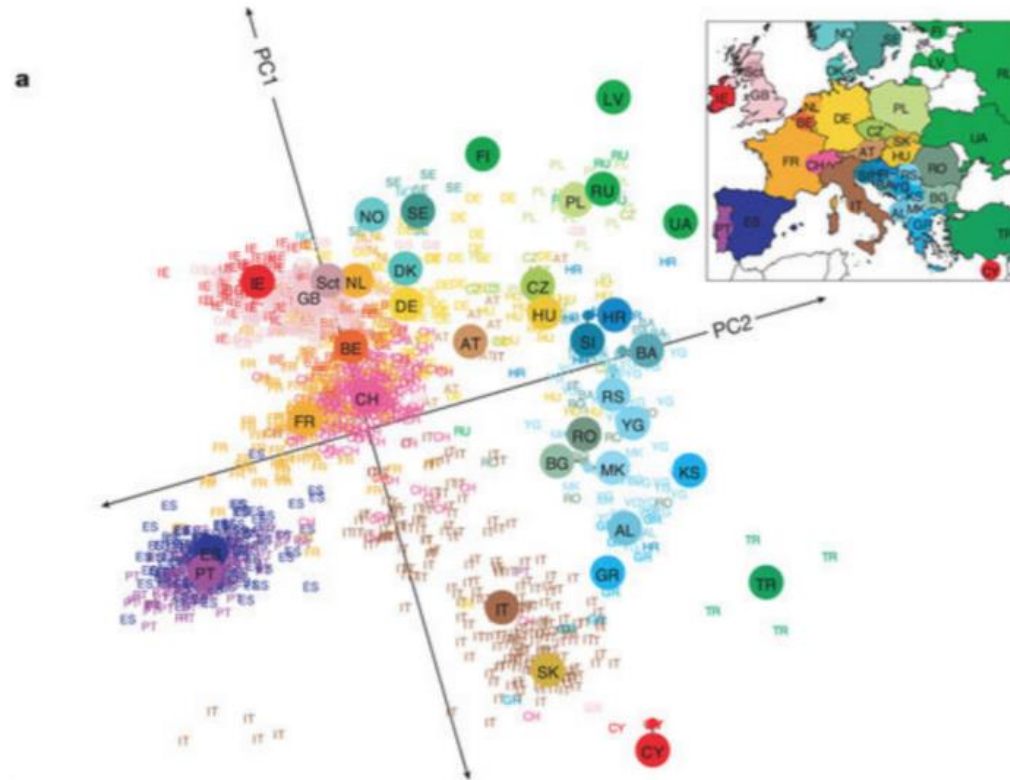


Figure 6: Plot from [1], depicting genomes for 1387 Europeans projected onto top 2 principal components. Colors/labels of datapoints correspond to geographic location of the individuals. Map of Europe (with same coloring) included in upper right for reference.

The classics: PCA

Goal: find a k -dimensional (linear) subspace explaining most of the variance in the data.

Assume the data is centered, that is $\mathbb{E}[x] = 0$

Warmup: let $k=1$.

$$\max_{\{v: \|v\|=1\}} \frac{1}{m} \sum_{\text{samples } x_i} \langle v, x_i \rangle^2$$

Variance: $\mathbb{E}[\langle v, x \rangle^2]$



The classics: PCA

Goal: find a k-dimensional (linear) subspace explaining most of the variance in the data.

$$\max_{\{v_1, v_2, \dots, v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} (\text{length of } x_i \text{ on } \text{span}(v_1, v_2, \dots, v_k))^2$$

$$= \max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} \sum_{j=1}^k \langle x_i, v_j \rangle^2$$

Variance: $\mathbb{E}[\sum_j \langle v_j, x \rangle^2]$



The classics: PCA

How to do this efficiently?

$$\max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} \sum_{j=1}^k \langle x_i, v_j \rangle^2$$

A convenient rewrite:

$$= \max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} \sum_{j=1}^k (v_j^T x_i)(x_i^T v_j)$$

$$= \max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} \sum_{j=1}^k v_j^T (x_i x_i^T) v_j$$

$$= \max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \sum_{j=1}^k v_j^T \left(\frac{1}{m} \sum_{\text{samples } x_i} (x_i x_i^T) \right) v_j$$

The classics: PCA

How to do this efficiently? – **Singular Value Decomposition!!**

$$\max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} \sum_{j=1}^k \langle x_i, v_j \rangle^2$$

A convenient rewrite:

$$\begin{aligned} &= \max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \sum_{j=1}^k v_j^T \underbrace{\left(\frac{1}{m} \sum_{\text{samples } x_i} (x_i x_i^T) \right)}_D v_j \\ &= \max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \sum_{j=1}^k v_j^T D v_j \end{aligned}$$

If D is **diagonal** (w/ positive entries): easy to see max is $\sum_{j=1}^k D_{jj}$

$$\cdot \sum_{j=1}^k v_j^T D v_j = \sum_j \sum_i (v_j)_i^2 D_{ii} \leq \sum_{j=1}^k D_{jj}, \text{ as } (v_j)_i^2 = 1$$

The classics: PCA

How to do this efficiently? – **Singular Value Decomposition!!**

$$\max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \frac{1}{m} \sum_{\text{samples } x_i} \sum_{j=1}^k \langle x_i, v_j \rangle^2$$

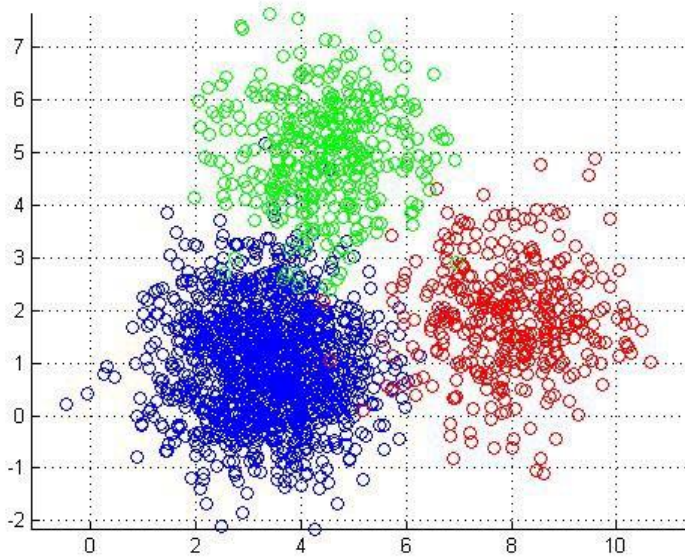
A convenient rewrite:

$$\begin{aligned} &= \max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \sum_{j=1}^k v_j^T \underbrace{\left(\frac{1}{m} \sum_{\text{samples } x_i} (x_i x_i^T) \right)}_D v_j \\ &= \max_{\text{orthonormal } \{v_1, v_2, \dots, v_k\}} \sum_{j=1}^k v_j^T D v_j \end{aligned}$$

If D is **diagonal** (w/ positive entries): easy to see max is $\sum_{j=1}^k D_{jj}$

O/w: reduce to diagonal case by taking SVD of $D = U \tilde{D} U^T$ and reducing to diagonal case: easy to see max is $\sum_{j=1}^k \lambda_j(D)$

The classics: clustering

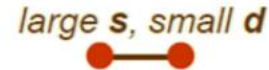


Goal: group the data into clusters of nearby points.

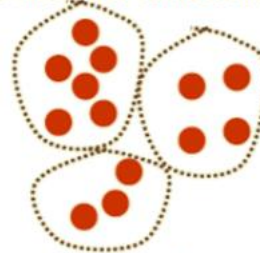
What's needed for clustering?

1. Proximity measure, *either*

- similarity measure $s(x_i, x_k)$: large if x_i, x_k are similar
- dissimilarity (or distance) measure $d(x_i, x_k)$: small if x_i, x_k are similar



2. Criterion function to evaluate a clustering



good clustering



bad clustering

3. Algorithm to compute clustering

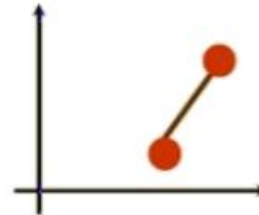
The classics: clustering

Popular distance metrics:

- Euclidean distance

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^d (\mathbf{x}_i^{(k)} - \mathbf{x}_j^{(k)})^2}$$

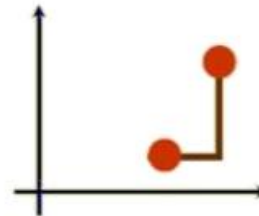
- translation invariant



- Manhattan (city block) distance

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^d |\mathbf{x}_i^{(k)} - \mathbf{x}_j^{(k)}|$$

- approximation to Euclidean distance, cheaper to compute



- They are special cases of **Minkowski distance**:

$$d_p(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^m |\mathbf{x}_{ik} - \mathbf{x}_{jk}|^p \right)^{\frac{1}{p}}$$

(p is a positive integer)

The classics: clustering

Criterion functions:

Intra-cluster cohesion

- Cohesion measures how near the data points in a cluster are to the cluster “center”.

Inter-cluster separation

- Separation means that different cluster centroids should be far away from one another.

In most applications, expert judgments are still the key

The classics: clustering

How many clusters?



- Possible approaches
 1. fix the number of clusters to k
 2. find the best clustering according to the criterion function (number of clusters may vary)

K-means clustering

If the distance metric is the **Euclidean distance**, and the measure of cohesion is the **average distance from the centroid**: we get the **k-means objective**.

$$\operatorname{argmin}_{\{r_{nk}, \mu_k\}} \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \mu_k\|^2$$

Is point n in cluster k ?

Centroid of k -th cluster

K-means clustering

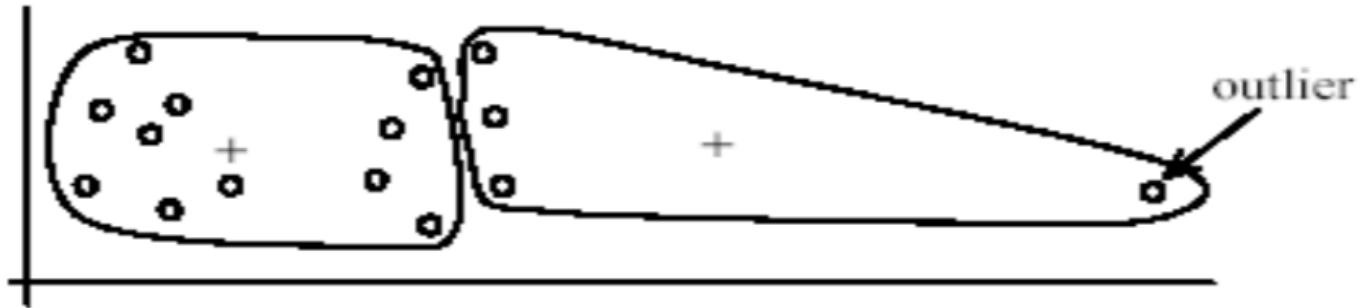
A natural iterative algorithm, which as we will see later is a variant of the **EM (expectation-maximization)** algorithm:

```
Input: Data set  $X = \{x^{(1)}, x^{(2)}, \dots, x^{(m)} | x^{(i)} \in \mathbb{R}^n\}$ 
Output: Cluster centroids  $\mu_{i=1, \dots, k} \in \mathbb{R}^n$ ; Cluster assignments  $c \in \mathbb{Z}$ 
1 Initialize  $k$  cluster centroids  $\mu_1, \dots, \mu_k \in \mathbb{R}^n$  randomly from  $X$ ;
2 repeat
3   for  $i = 1, \dots, m$  do // Update cluster assignments
4     | set  $c^{(i)} = \arg \min_j \|x^{(i)} - \mu_j\|^2$ ;
5   end
6   for  $j = 1, \dots, k$  do // Update cluster centroids
7     | set  $\mu_j = \frac{\sum_{i=1}^m 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m 1_{\{c^{(i)}=j\}}}$ ;
8   end
9 until Convergence;
10 return  $\mu$  and  $c$ ;
```

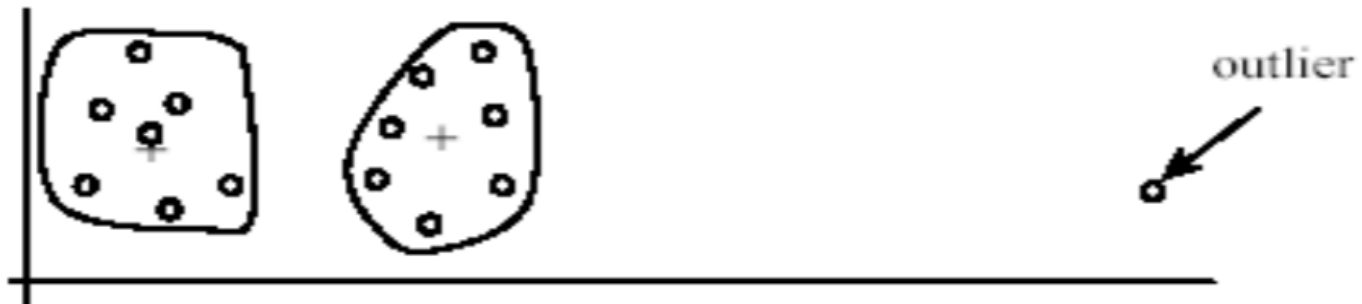
Algorithm 1: Algorithm of batch-version for K-means

Some weaknesses

Very sensitive to outliers:



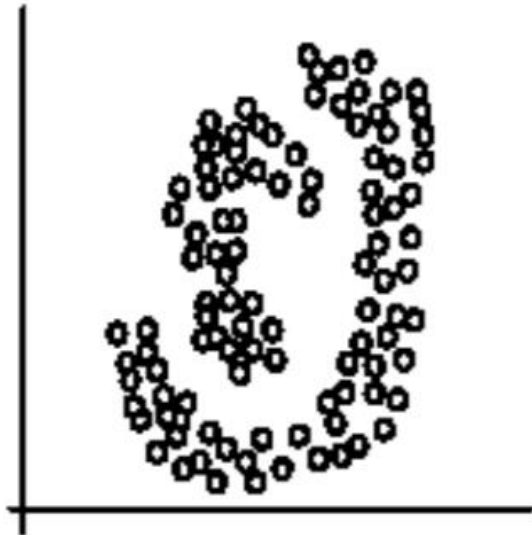
(A): Undesirable clusters



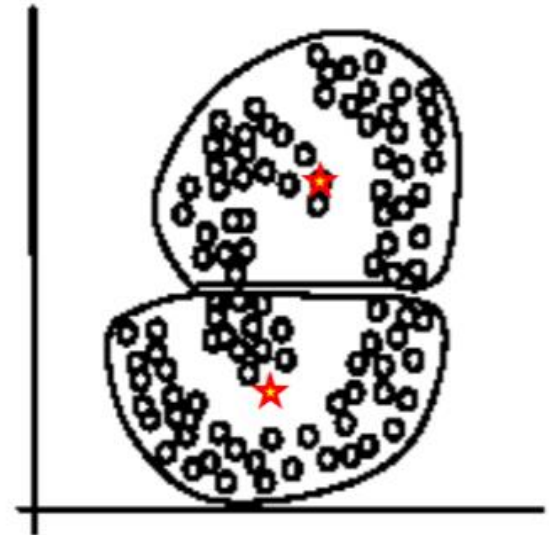
(B): Ideal clusters

Some weaknesses

Not suitable for non-spherical clusters:



(A): Two natural clusters



(B): k -means clusters