Lec 13 [Andrew NG]

Key ideas

1. Diagnostics for debugging learning algos
2. Error and ablative analyses
3. How to get started on a ML problem: - premature stat. optimisation.

Ex:

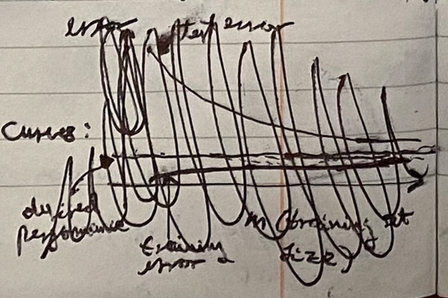
- Anti-spam: carefully choose 100 words to use as features (instead of all ~50k)
- Logistic regr. + regularisation (= Bayesian logregr) w/ grad-ascent gets 20% train error: too high. [$\max_\theta \sum_{i=1}^{m} \log p(y^i|x^i,\theta) - \lambda|\theta|^2$]
- What next?

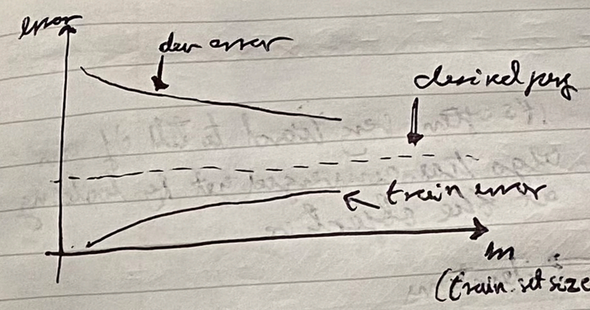
- Common approach: Try improving algo in various ways.
  - Try getting more data & hope.
  - Try smaller set of features
  - Try more features
  - Try changing the features. Eg email header vs body features
  - Run gradient ascent for more iterations
    - or Newton's method
    - or use different λ
  - Try using SVM

Just brainstorming is already ahead of many teams. What diagnostics to run to pick best approach?

Well, suppose you suspect the problem is either overfitting (high variance) or too few features (high bias). ⇒ Diagnostic: Variance: test error ⇒ 20%.
While of bias: test error ~ train error

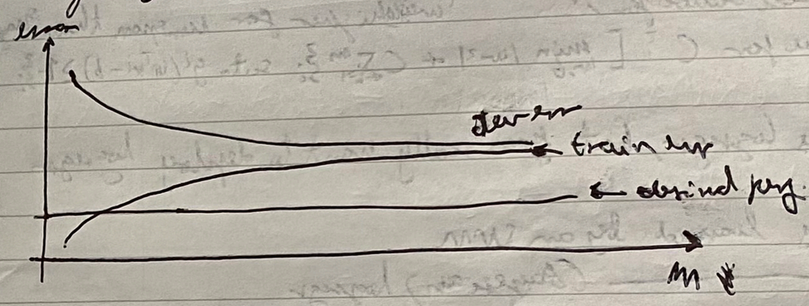
Typically
High variance often gives such learning curves:





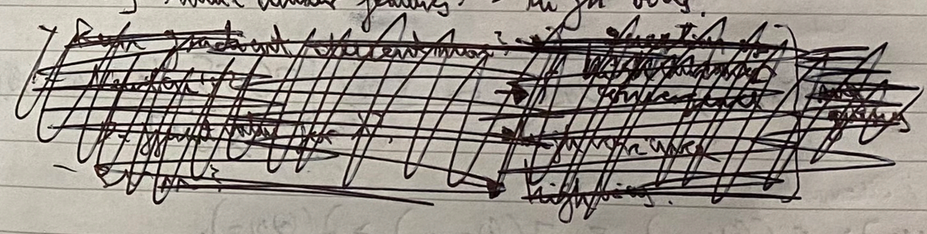
Note the large dev-train error gap for this high variance curve

High bias:



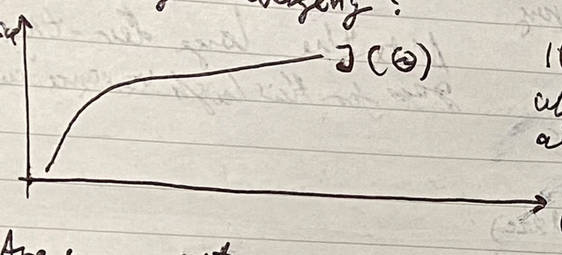
Note: small gap & even train error too high

Fixes to try:
- More training examples? → No, high variance
- Smaller set of features? → ——
- Larger set of features? → high bias, yes.
- Try email header features? → high bias.



- Bias vs variance is one common diagnostic
- For other problems, it's up to your ingenuity to find your own diagnostic to figure out what's wrong.
- Another example:
  - Logreg gets 2% error on both spam & non-spam.
  - SVM w/ linear kernel gets 10% error on spam, and .01% error on non-spam
  - But you want to use logreg for computational efficiency etc.
- What to do next?

- Other common Qestions:
  - is the algo converging?



It's often very hard to tell if an algo has converged just by looking at the objective

  - Are you optimising the right gain function?
  - Ie, what you care about: $a(\theta) = \sum_i w^{(i)} 1\{h_\theta(x^{(i)}) = y^{(i)}\}$ ↑ weighting higher for non-spam than spam
  - Logistic regr? Correct value for λ?
  - SVM? Correct value for C? [$\min_{w,b} \|w\|^2 + C\sum_{i=1}^m \xi_i$ s.t. $y^{(i)}(w^T x^{(i)} - b) \geq 1 - \xi_i$

summary:
An SVM outperforms logregr, but you really want to deploy logregr for your application.
Let θ_{SVM} be parameters learned by an SVM
Let θ_{BLR} be — .. ———— (Bayesian) logregr
You care about weighted acc $a(\theta) = \max_\theta \sum_i w^{(i)} 1\{h_\theta(x^{(i)}) = y^{(i)}\}$, and (as mentioned) $a(\theta_{SVM}) > a(\theta_{BLR})$

Two hypotheses: (1) Grad. ascent simply isn't correctly optimising $J(\theta)$, or (2) $J(\theta)$ [ie, model] is a bad gain function [model inappropriate eg λ too large]

Diagnostics: Is $J(\theta_{SVM}) > J(\theta_{BLR})$, or $\leq$?? [$J(\theta) = \sum_{i=1}^m \log p(y^{(i)}|x^{(i)};\theta) - \lambda\|\theta\|^2$]

Case 1:
$a(\theta_{SVM}) > a(\theta_{BLR})$ & $J(\theta_{SVM}) > J(\theta_{BLR})$.
But: BLR was trying to maximise $J(\theta)$ ⇒ it fails to converge.
Problem lies in optimisation algorithm.

Case 2:
$a(\theta_{SVM}) > a(\theta_{BLR})$
$J(\theta_{SVM}) \leq J(\theta_{BLR})$
This means J is correctly maximised, but that $J(\theta)$ is the wrong gain function to use as SVM does better on weighted accuracy $a(\theta)$.

⇒ problem is with objective function itself.

- Fixes to try: (Coted)
  - Run gradient descent for more iterations: → fixes optimisation algo.
  - Try Newton's method: .. ..
  - Use a different λ: → fixes optimisation objective
  - Try using an SVM: → .. ..

Machine Learning Algorithm: Stanford Helicopter Project

1. Build a simulator of helicopter
2. Choose a cost function. Say $J(\theta) = \|x - x_{desired}\|^2$ (x = helicopter pos.)
3. Run reinforcement learning (RL) algorithm to fly helicopter in sim., so as to try to minimise $\theta_{RL} = \arg\min_\theta J(\theta)$

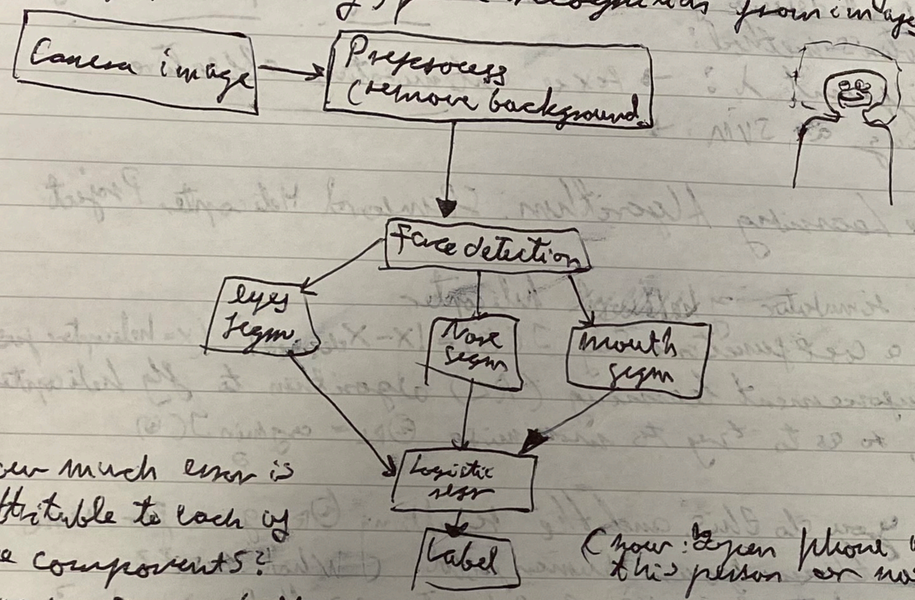
Suppose you do this, and the resulting θ_{RL} gives far worse performance than your human pilot. What next?
- Improve simulator? Modify cost function J? Modify RL algo?

Ok. Well, suppose that: 1. The helicopter sim. is accurate. 2. The RL algo correctly controls the heli in sim so as to minimise $J(\theta)$ 3. Minimising $J(\theta)$ corresponds to correct autonomous flight
Then: The learned parameters θ_{RL} should fly well on the actual helicopter.

Diagnostics:
1. If θ_{RL} flies well in sim but not real life ⇒ simulator is the probl.
   Otherwise:
2. Let θ_{human} be the human control policy. If $J(\theta_{human}) \geq J(\theta_{RL})$, then the problem is in the reinforcement learning algo. (failing to minimise the cost function J.)
3. If $J(\theta_{human}) < J(\theta_{RL})$, then the problem is in the cost function. (Maximising it doesn't correspond to good autonomous flight).

Error Analysis: explain difference between where we are, and ~~perfection~~

Many applications combine many different learning components into a "pipeline". Eg, face recognition from images: [artificial ex]



(Your: person wearing glasses or not?)

How much error is attributable to each of the components?

Plug in ground-truth for each component, and see how acc. changes.

orig.	Component	Accuracy
	Overall sys.	85%
adding ground truths (accumulating)	Preprocess	~~+1%~~ ~~85~~ ~~86~~ ~~85.1%~~ → 85.1%
	Face det.	~~+6%~~ → 91%
	Eyes segm.	~~+2%~~ → 93%
	Nose ...	~~+1%~~ → 96%
	Mouth ...	→ 97%
	Log. regr	→ 100%

Conclusion: in this example, most room for improvement in face detection and eyes segm.

Nice way to isolate different factors!

Tips: whenever you find weird/unexpected behaviour → never ignore it's gold!

Ablative analysis: explain diff. between where you are and something much worse.

=> Remove components one at a time to see how it breaks.

Ex. for spam classifier:

Component	Accuracy
Overall sys.	99.9%
Spelling corr.	99%
Under host feats.	98.9%
Email header feat.	98.9%
Email text parser feats	95%
Javascript parser	99.5%
Features from imgs.	94%

← Cumulative removal (in this case, but in gen. also possible)

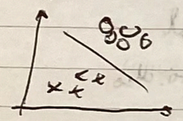
Conclusion: The email text parser feats. account for most of the improvement.

[redacted]

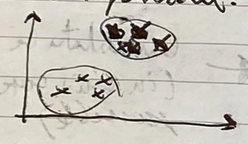
Lecture 14

Topic:
Unsupervised learning
- k-means clustering
- Mixture of Gaussians
- Expectation maximization (EM) algorithm & its derivation

Supervised:



Unsupervised:



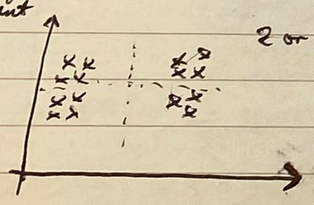
(unlabelled)
Goal: find interesting properties about data

k-means clustering algo:
Data $\{x^1, \dots, x^m\}$

1. Initialize cluster centroids $\mu_1, \dots \mu_k \in \mathbb{R}^n$ randomly

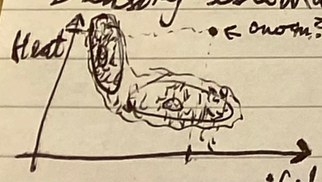
2. Repeat until Convergence:
   a) Set $c^i := \operatorname{argmin}_j |x^i - \mu_j|^2$ ("closest the points")
   b) For $j = 1, \dots, k$
   $$\mu_j := \frac{\sum_{i=1}^m 1\{c^i = j\} x^i}{\sum_{i=1}^m 1\{c^i = j\}}$$

$J(c, \mu) = \sum_{i=1}^m |x^i - \mu_{c^i}|^2$ ← This cost function is strictly decreasing
(c: assignment, μ: centroid)



2 or 4 clusters?

Density estimation

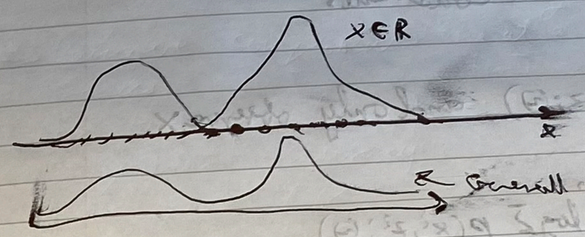


Anomaly detection

Model $p(x) < \varepsilon \rightarrow$ anomaly!
$> \varepsilon \rightarrow$ looks ok

Looks like data comes from two Gaussians
⇒ Let's use the "mixture of Gaussians" method.

1D example



Mixture of Gaussians model:
Suppose there's a latent (hidden/unobserved) random variable z, and x^i, z^i are distributed $p(x^i, z^i) = p(x^i | z^i) p(z^i)$, where
$z^i \sim \text{Multinomial}(\phi)$, $z \in \{1, \dots, k\}$
$(x^i | z^i = j) \sim \mathcal{N}(\mu_j, \Sigma_j)$

If we ~~need~~ know z^i's, can use MLE: $\ell(\phi, \mu, \Sigma) = \sum_{i=1}^m \log p(x^i, z^i; \phi, \mu, \Sigma)$

$\Rightarrow \phi_j = \frac{1}{m} \sum_{i=1}^m 1\{z^i = j\}$ & $\mu_j = \sum_{i=1}^m 1\{z^i = j\} x^i \Big/ \sum_{i=1}^m 1\{z^i = j\}$ & $\Sigma = \dots$ (see lecture notes)

EM:
E-step: (Guess value of z^i's): Set $w_j^i = p(z^i = j | x^i; \phi, \mu, \Sigma)$
$= p(x^i | z^i = j) p(z^i = j) \Big/ \sum_{\ell=1}^k p(x^i | z^i = \ell) p(z^i = \ell)$
(this as numerators)

$\mathcal{N}(\mu_j, \Sigma_j) = \frac{1}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(x^i - \mu_j)^T \Sigma_j^{-1} (x^i - \mu_j)\right)$ ← $\phi_j, z \sim$ multinomial(ϕ)

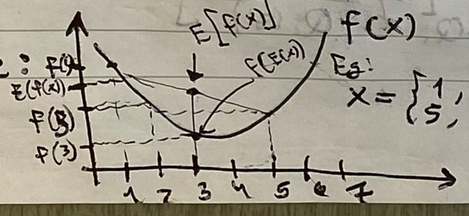
M-step: $\phi_j := \frac{1}{m} \sum_{i=1}^m w_j^i$ & $\mu_j = \sum_{i=1}^m w_j^i x^i \Big/ \sum_{i=1}^m w_j^i$

($1\{z^i = j\} \rightarrow w_j^i = E[1\{z^i = j\}]$)

Jensen's inequality

Let f be a convex function ($f'' \geq 0$) & X be a RV; then $f(E(X)) \leq E[f(X)]$.

Example:



Ex: $X = \begin{cases} 1, & \text{prob} = 1/2 \\ 5, & \text{prob} = 1/2 \end{cases}$

$\rightarrow E(f(x)) = \frac{1}{2}[f(1) + f(5)]$
$= 3 \rightarrow f(3)$

Further, if $f''(x) > 0$ (f is strictly convex), then

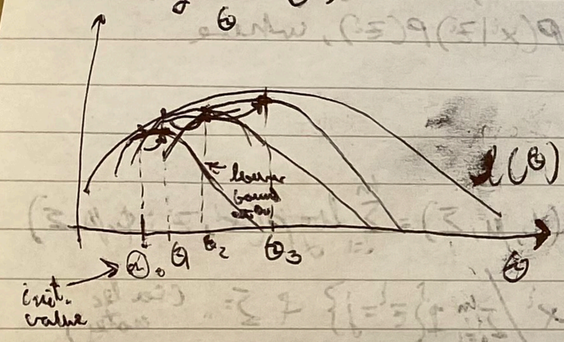
$E[f(x)] = f(E(x))$ only if x is a constant.

Have a model for $p(x, z; \theta)$, and only observe x.

$\{x^1, \dots, x^m\}$

$\ell(\theta) = \sum_{i=1}^m \log p(x^i; \theta) = \sum_{i=1}^m \log \sum_{z^i} p(x^i, z^i; \theta)$

Want: $\arg\max_\theta \ell(\theta)$



$\max_\theta \sum_i \log p(x^i; \theta) = \sum_i \log \sum_{z^i} p(x^i, z^i; \theta)$

$= \sum_i \log \sum_{z^i} Q_i(z^i) \left[\frac{P(x^i, z^i; \theta)}{Q_i(z^i)}\right]$, where $Q_i(z^i)$ is a pdf ($\sum_i Q_i = 1$)

$= \sum_i \log E_{z^i \sim Q_i} \left[\frac{P(x^i, z^i; \theta)}{Q_i(z^i)}\right]$

Jensen's ineq. ($f E x \le E f x$) $\Rightarrow \le \sum_i E_{z^i \sim Q_i} \left[\log \frac{P(x^i, z^i; \theta)}{Q_i(z^i)}\right] = \sum_i \sum_{z^i} Q_i(z^i) \log \frac{P(x^i, z^i; \theta)}{Q_i(z^i)}$

On a given iteration of EM (with parameters θ), we want:

$\log E_{z^i \sim Q_i} \left[\frac{P(x^i, z^i; \theta)}{Q_i(z^i)}\right] = E_{z^i \sim Q_i} \left[\log \frac{P(x^i, z^i; \theta)}{Q_i(z^i)}\right]$

For this to hold, we need $\frac{P(x^i, z^i)}{Q_i(z^i)} = \text{Constant}$

Set $Q_i(z^i) \propto P(x^i, z^i; \theta)$, $Q(z^i) = P(x^i; z^i; \theta)$

$\sum_{z^i} Q_i = 1$, $Q_i(z^i) = \frac{P(x^i, z^i; \theta)}{\sum_{z^i} P(x^i, z^i; \theta)} = \dots = p(z^i | x^i; \theta)$

E-step: set $Q_i(z^i) = p(z^i | x^i; \theta)$ ← w_j^i

M-step: $\theta_k = \arg\max_\theta \sum_i \sum_{z^i} Q_i(z^i) \log \frac{P(x^i, z^i; \theta)}{Q_i(z^i)}$

Lecture 15

Outline:
- EM Convergence
- Gaussian properties
- Factor analysis
- Gaussian marginals & cond. forms
- EM steps

Recap:

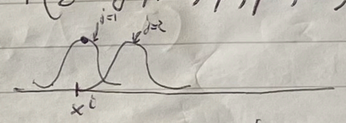
E-step: $Q_i(z^i) := P(z^i | x^i; \theta)$

M-step: $\theta := \arg\max_\theta \sum_i \sum_{z^i} Q_i(z^i) \log \frac{P(x^i, z^i; \theta)}{Q_i(z^i)}$

$P(x^i, z^i) = P(x^i | z^i) P(z^i)$, $z^i \sim \text{Multinomial}(\phi)$ $[P(z^i = j) = \phi_j]$

$x^i | z^i = j \sim \mathcal{N}(\mu_j, \Sigma_j)$

E-step: $Q_i(z^i = j)$ "$P(z^i = j)$" $= P(z^i = j | x^i; \phi, \mu, \Sigma) = w_j^i$



M-step: $\max_{\phi,\mu,\Sigma} \sum_i \sum_{z^i} Q_i(z^i) \log \frac{P(x^i, z^i; \phi, \mu, \Sigma)}{Q_i(z^i)}$ ← $P(z^i)$

$= \sum_i \sum_j w_j^i \log \frac{(w_j^i)^{-1}}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \underbrace{\exp\left[-\frac{1}{2}(x^i - \mu_j)^T \Sigma_j^{-1} (x^i - \mu_j)\right]}_{P(x^i | z^i; \ldots)} \phi_j$

$\nabla_{\mu_j}(\ldots) = 0 \Rightarrow \mu_j = \frac{\sum_i w_j^i x^i}{\sum_i w_j^i}$ ← weight of x^i @ Gaussian j

$\nabla_\phi(\ldots) = 0 \Rightarrow \phi_j = \frac{\sum_i w_j^i}{\sum_i \sum_j w_j^i}$

$\nabla_{\Sigma_j}(\ldots) = 0$

Note: Equivalent interpp:

$J(\theta, Q) = \sum_i \sum_{z^i} Q_i(z^i) \log \frac{P(x^i, z^i; \theta)}{Q_i(z^i)}$

We know $\ell(\theta) \geq J(\theta, Q)$ for any θ, Q (Jensen's inequality)

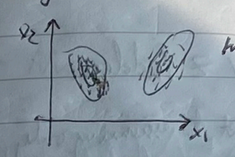
E-step: Maximise J wrt Q

M-step: Maximise J wrt θ

Factor Analysis Algorithm

Mixture of Gaussians:

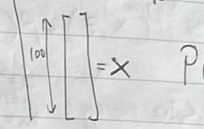
Say $n = 2$, $m = 100$



$m \gg n$

$m \approx n$, or maybe $m < n$

Say $m = 30$, $n = 100$, eg. (eg 30 days, 100 sensors)

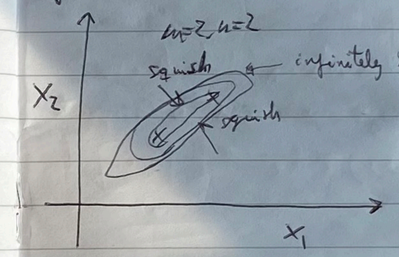


$P(X) \leq \varepsilon \Rightarrow$ Anomaly!

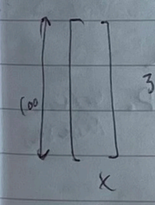
Model as single Gaussian: $X \sim \mathcal{N}(\mu, \Sigma) \Rightarrow$ MLE: $\mu = \frac{1}{m}\sum_i x^i$

$\Sigma = \frac{1}{m}\sum_i (x^i - \mu)(x^i - \mu)^T$

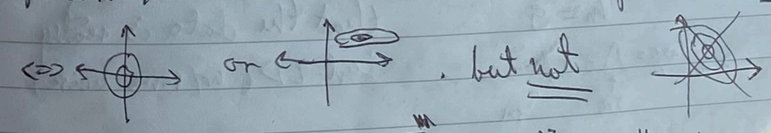
If $m < n$, then Σ will be singular, but $\mathcal{N}(\mu, \Sigma) \sim \frac{\exp[-(\ldots)\Sigma^{-1}(\ldots)]}{(2\pi)^{n/2}|\Sigma|^{1/2}}$ ← $=0$!!! ← underlined



100 psychological attributes; 30 people



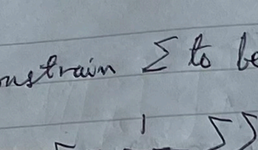
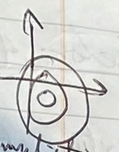
Option 1: constrain Σ to be diagonal: $\Sigma = \begin{bmatrix} \sigma_1^2 & & 0 \\ & \sigma_2^2 & \\ 0 & & \ddots \end{bmatrix}$



$\Rightarrow$ MLE gives $\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^i - \mu_j)^2$ for all n params $(j = 1, 2, \ldots, n)$

But assuming everything is uncorrelated isn't exactly smart...

Bad model, usually.

Option 2: Constrain Σ to be $\Sigma = \sigma^2 \mathbb{1}$ ⇔  or 

~~Easier to use~~

⇒ MLE ~~bbb~~: $\sigma = \frac{1}{mn} \sum_i \sum_j (x_j^i - \mu_j)^2$. ~~Very bad~~ simplistic model.

Factor Analysis model: $P(x,z) = P(x|z)P(z)$ w/ z hidden.

$z \sim \mathcal{N}(0, \mathbb{1})$, $z \in \mathbb{R}^d$, $d < n$. eg: $d=3$, $n=100$, $m=30$

Equivalent to assuming only d factors are actually needed to accurately model the data

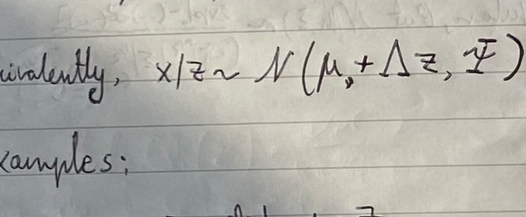
Assume $x = \mu + \Lambda z + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \Psi)$

Parameters: $\mu \in \mathbb{R}^n$, $\Lambda \in \mathbb{R}^{n \times d}$, $\Psi \in \mathbb{R}^{n \times n}$ & diagonal.

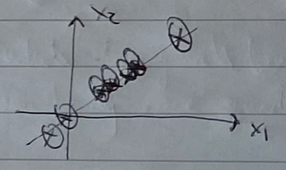
Equivalently, $x|z \sim \mathcal{N}(\mu + \Lambda z, \Psi)$.

Examples:

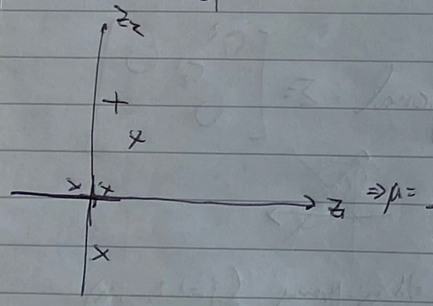
$z \in \mathbb{R}^1, x \in \mathbb{R}^2 \Rightarrow d=1$ & $n=2$ & $m=7$



Say $\Lambda = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Rightarrow \Lambda z + \mu \in \mathbb{R}^2$. And $\Psi = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$



⇒ Here we basically assume the data is ~~actually~~ 1D. (w/ some noise)



$z \in \mathbb{R}^2 \Leftrightarrow d=2$

$x \in \mathbb{R}^3 \Leftrightarrow n=3$

$m = 5$

$\Rightarrow \mu = \underset{3\times 2}{\Lambda} \underset{2\times 1}{z} + \underset{3\times 1}{\mu}$

Here we're assuming that the data is really just 2D (w/ some noise)

Multivariate Gaussian

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{matrix} \updownarrow r \\ \updownarrow s \end{matrix}$ $x_1 \in \mathbb{R}^r, x_2 \in \mathbb{R}^s \Rightarrow x \in \mathbb{R}^{r+s}$

$x \sim \mathcal{N}(\mu, \Sigma) \Rightarrow \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$ & $\Sigma = \begin{bmatrix} \overset{r}{\Sigma_{11}} & \overset{s}{\Sigma_{12}} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \begin{matrix} \updownarrow r \\ \updownarrow s \end{matrix}$

- Marginal: $p(x_1) = ?$ Well, $p(x) = p(x_1, x_2)$ and $\int_{x_2} p(x_1, x_2) dx_2 = p(x_1)$.

If $\frac{\exp\left(\begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix}^T \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}^{-1} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix}\right)}{(2\pi)^{n/2} |\Sigma|^{1/2}} = p(x_1, x_2)$, then it turns out that $x_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$

- Conditional: $p(x_1|x_2) = ?$ Well, $p(x_1, x_2)/p(x_2)$ and $x_1|x_2 \sim \mathcal{N}(\mu_{1|2}, \Sigma_{1|2})$

w/ $\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$

&

$\Sigma_{1|2} = \Sigma_{22} - \Sigma_{12}^T \Sigma_{22}^{-1} \Sigma_{21}$

Now we're ready to derive the EM algo

1. Derive $p(x,z)$:

$\begin{pmatrix} z \\ x \end{pmatrix} \sim \mathcal{N}(\mu_{x,z}, \Sigma)$ & $\begin{bmatrix} \text{recall } x = \mu + \Lambda z + \varepsilon \\ z \sim \mathcal{N}(0, \mathbb{1}) \end{bmatrix} \Rightarrow \begin{matrix} E[z] = 0 \\ E[x] = \mu \end{matrix}$

$\Rightarrow \mu_{x,z} = \begin{bmatrix} 0 \\ \mu \end{bmatrix} \begin{matrix} \updownarrow d \\ \updownarrow n \end{matrix}$, $\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} = \begin{bmatrix} E[(z-0)(z-0)^T] & E[(z-0)(x-\mu)^T] \\ E[(x-\mu)(z-0)^T] & E[(x-\mu)(x-\mu)^T] \end{bmatrix}$

$\overset{d}{\leftrightarrow}$ $\overset{n}{\leftrightarrow}$

$\Sigma_{22} = E[(x-\mu)(x-\mu)^T] = E[(\Lambda z + \mu + \varepsilon - \mu)(\Lambda z + \mu + \varepsilon - \mu)^T]$

$= E[\Lambda z z^T \Lambda^T + \Lambda z \varepsilon^T + \varepsilon z^T \Lambda^T + \varepsilon \varepsilon^T] = \Lambda \underbrace{E[z z^T]}_{=\mathbb{1}} \Lambda^T + \Psi = \Lambda\Lambda^T + \Psi$

$\vdots$

$\Rightarrow \Sigma = \begin{bmatrix} \mathbb{1} & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{bmatrix}$

$$\Rightarrow \begin{bmatrix} z \\ x \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \vec{0} \\ \mu \end{bmatrix}, \begin{bmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{bmatrix} \right)$$

E-step:

Compute:

$Q_i(z^i) = p(z^i | x^i; \theta)$

$z^i | x^i \sim \mathcal{N}(\mu_{z^i|x^i}, \Sigma_{z^i|x^i})$, where $\mu_{z^i|x^i} = \vec{0} + \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}(x - \mu)$

$\Sigma_{z^i|x^i} = I - \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}\Lambda$

M-step:

We know that $Q_i(z^i) = \frac{\exp(\cdots)}{(2\pi)^{d/2}|\cdots|}$. (with the actual derivation (which isn't covered fully here), one needs to compute terms like $\int_{z^i} Q_i(z^i) z^i dz^i = \int_{z^i} \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(\cdots)) z^i dz^i = \underset{z^i \sim Q_i}{E}[z^i]$

$= \mu_{z^i|x^i}$

$\theta := \arg\max_\theta \sum_i \int_{z^i} Q_i(z^i) \log \frac{P(x^i, z^i)}{Q(z^i)} dz^i = \sum_i E_{z^i \sim Q_i}\left[\log \frac{P(x^i, z^i)}{Q_i(z^i)} \right]$

Plug in Gaussian density

ends up being quadr.

This that whole thing will be quadr, and when doing $\nabla_{\mu^{(i)}}$ you'll only face quadratic complexity.

Independent Components Analysis (Lec 16)

- CDFs
- ICA model

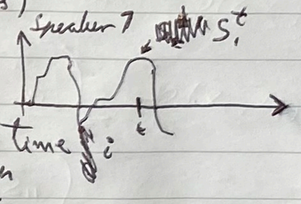
Reinforcement learning

- MDP (Markov decision process)

Sources: $s \in \mathbb{R}^n$

s_j^i = speaker j at time i

$x^i = As^i$, $x \in \mathbb{R}^n$

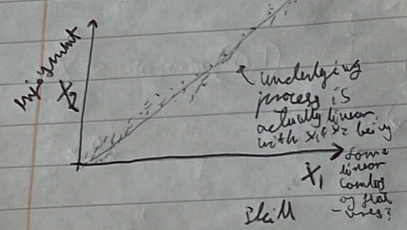


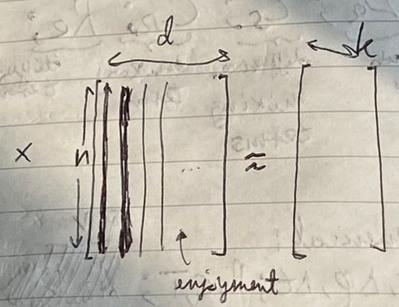
Goal: Find $w = A^{-1}$, so $s^i = wx^i$ | $w = \begin{bmatrix} — w_1^T — \\ \vdots \\ — w_n^T — \end{bmatrix}$

Principal & Independent CA (lec 15.5)

PCA

$\{x^i\}_{i=1}^n$, $x^i \in \mathbb{R}^d$



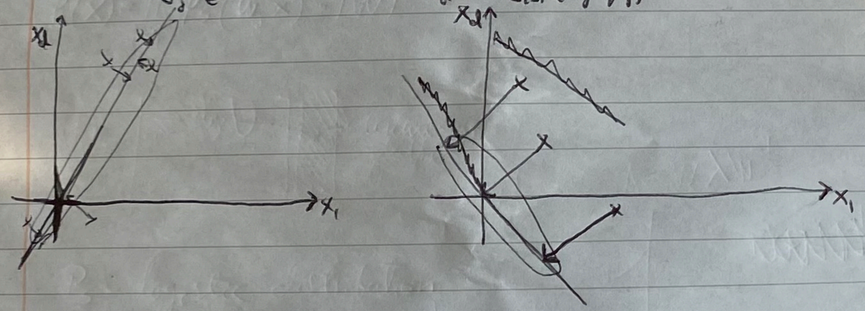


Standardize

$x_j^i \leftarrow \frac{x_j^i - \mu_j}{\sigma_j}$ ← mean & std dev of jth column.

$\mu_j = \frac{1}{n}\sum_{i=1}^n x_j^i$

$\sigma_j^2 = \frac{1}{n}\sum_{i=1}^n (x_j^i - \mu_j)^2$



$u \in \mathbb{R}^d$ is unit length

$\underbrace{\text{Proj}(u)}_{\frac{uu^T}{u^Tu=1}} \vec{x} = \langle \vec{x}^i, \vec{u} \rangle \vec{u}$

$\Rightarrow$ Want to find u s.t. $\frac{1}{n}\sum_{i=1}^n \|\text{Proj}(u)x\|_2^2 = \frac{1}{n}\sum_{i=1}^n \|\langle x^i, u\rangle u\|_2^2 = \frac{1}{n}\sum_{i=1}^n (\langle x^i, u\rangle)^2 = u^T\underbrace{\left(\frac{1}{n}\sum_{i=1}^n x^i x^{iT}\right)}_{\text{= sample covariance matrix (since } x \text{ has been standardized).}} u$ is maximal.

$u = \underset{u}{\text{argmax}}\; u^T\left[\frac{1}{n}\sum_{i=1}^n x^i x^{iT}\right]u$. $\underset{u}{\text{argmax}}\; u^TAu \Rightarrow$ eigenvector of largest eigenvalue of A.

sorted in descending order & these are all positive due to XX^T being symmetric

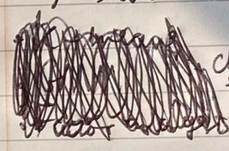
Now, we find k s.t. $\frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^d \lambda_i} = $ eg 95% ($\Leftrightarrow$ "retained 95% of variance in data").

SVD-rmk: If X is square & symmetric, then X has an orthogonal eigenvector basis with real eigenvalues.

Eigendecomp on $X^TX \iff$ SVD on X

rmk: Two (equivalent) interps of PCA: (1) Hyperplanes that capture the most variance and (2) Hyperplanes that allow you to project the data on with least L2 error.

So far, we've seen 4 algos in unsupervised learning:



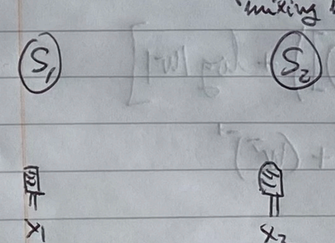
	non-probabilistic	probabilistic	
clustering subproblem algos	k-means	GMM	classification
	PCA	Factor analysis	regression problem

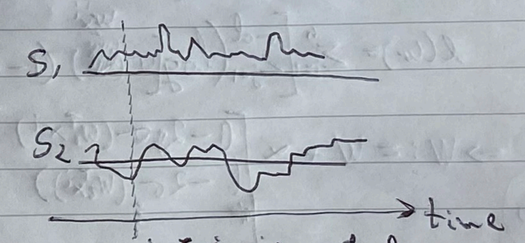
EM

ICA

Cocktail-party problem: d speakers & d microphones, $s \in \mathbb{R}^d$ is one instantaneous recording, $x \in \mathbb{R}^d$ & $x = As$, s_j^i ← (i = time, j = speaker) & x_j^i (i = time, j = microphone)

↑ mixing matrix

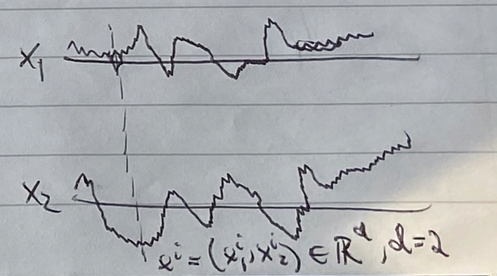




$x^i = As^i$

$w = A^{-1}$ [unmixing matrix]

$s^i \leftarrow wx^i$



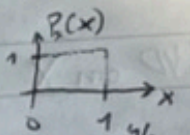
Assumptions: $*S = *X$

$S = WX$

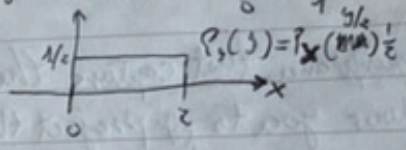
$S_j \perp S_k$, $j \neq k$. (Independence)

S_j is NOT Gaussian

$X \sim \text{uniform}(0,1)$



$y = 2x$



now $x \in \mathbb{R}^d$

$y = Wx$

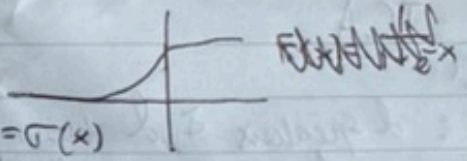
$\mathbb{R}^d \quad \mathbb{R}^{d \times d} \quad \mathbb{R}^d$

$P_y(y) = P_x(W^{-1}y) \frac{1}{|W|}$ ← Jacobian

$$P(x) = \prod_{j=1}^{d} P_S(w_j^T x) \cdot |W|$$

$$W = \begin{bmatrix} - w_i - \end{bmatrix}$$ unmixing matrix

If $P_S \sim$ logistic distribution



CDF of $F(x) = \frac{1}{1+e^{-x}} = \sigma(x)$

pdf of $f(x) = \sigma(x)(1-\sigma(x))$

$$l(w) = \sum_{i=1}^{n} \left[\sum_{j=1}^{d} \left(\log \left[\sigma(w_j^T x^i) \left(1 - \sigma(w_j^T x^i)\right) \right] \right) + \log |W| \right]$$

$$\Rightarrow W := W + \alpha \begin{bmatrix} (1 - 2\sigma(w_1^T x^i)) \\ (1 - 2\sigma(w_2^T x^i)) \\ \vdots \end{bmatrix} x^{iT} + (W^T)^{-1}$$

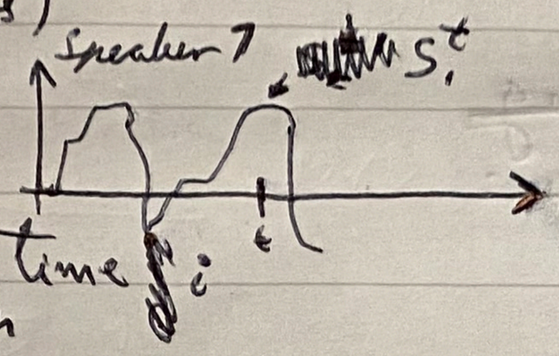
$\left(\frac{d \ln}{dx} = \sin(x)\right)$

Independent Components Analysis (Lec 16)

- CDFs
- ICA model

Reinforcement learning

- MDP (Markov decision process)



Sources: $s \in \mathbb{R}^n$

s_j^i = speaker j at time i

$x^i = As^i$, $x \in \mathbb{R}^n$

Goal: Find $w = A^{-1}$, so $s^i = wx^i$ | $w = \begin{bmatrix} — w_1^T — \\ \vdots \\ — w_n^T — \end{bmatrix}$

3 Definitions

1. CDF Cumulative distribution function

$$F(s) = P(S' = s) = \int_{-\infty}^{s} p_{S'}(s')\,ds' \Leftrightarrow p_{S'}(s) = \frac{d}{ds}F(s)$$

(value, RV)

- Plug in
- Gaussian
- density

2. And:

$$p_X(x) = p_{S'}(\underbrace{wx}_{=s}) \cdot \underbrace{|w|}_{\text{Jacobian}}$$

(determinant of w. needed to preserve norm.)

3. Finally: density of x

$p_{S'}(s) = ?$ Well, $F(s) = P(S' \le s) = (1+e^{-s})^{-1} = g(s)$ works well.

So $p_{S'}(s) = \frac{d}{ds}F(s)$.

$\underline{p(s) = \prod_{i=1}^{n} p_{S'}(s_i)}$, assume the n speakers speak independently

Now, $p_X(x) = p_{S'}(wx)\,|w| = \prod_{i=1}^{n} p_{S'}(w_i^T x)\,|w|$

$\Rightarrow$ MLE: $\ell(w) = \sum_{i=1}^{m} \log\left(\prod_{j=1}^{n} p_{S'}(w_j^T x^i)\,|w|\right)$. Can use stochastic gradient ascent here.

stoch. grad ascent:

$$\nabla_w \ell(w) = \begin{bmatrix} 1 - 2g(w_1^T x^i) \\ \vdots \\ 1 - 2g(w_n^T x^i) \end{bmatrix} x^{iT} + (w^T)^{-1}, \quad g(s) = (1+e^{-s})^{-1}$$

(sigmoid)

** This'll result in a good algorithm for unmixing the sources.

Reinforcement Learning

Chess playing program:

$R(s)$ = +1 for win (reward function)
state
−1 for loss
0 for draw

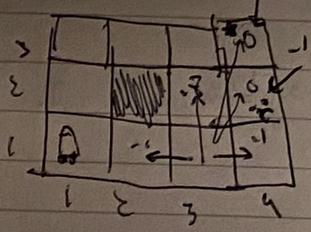
Credit assignment problem

Markov Decision Process (MDP):

$(S, A, \{P_{sa}\}, \gamma, R)$ — reward function

- S: set of states
- A: set of actions
- $\{P_{sa}\}$: state transition probabilities ($\sum_{s'} P_{sa}(s') = 1$)
- γ: discount factor, $\gamma \in [0,1)$

$s \to a$



This MDP has:

Actions: $\{N, S, E, W\}$

$P_{(3,1)N}((3,2)) = 0.8$

$P_{(3,1)N}((4,1)) = 0.1$

$P_{(3,1)N}((2,1)) = .1$

$P_{(3,1)N}(\text{rest}\ldots) = 0$

$R((4,3)) = +1$; $R((4,2)) = -1$; $R(s) = -.02$ (small negative r to punish lingering & loitering) for all other s.

---

s_0

Choose action a_0

Get to $s_1 \sim P_{s_0 a_0}$

Choose action a_1

Get to $s_2 \sim P_{s_1 a_1}$

$\vdots$

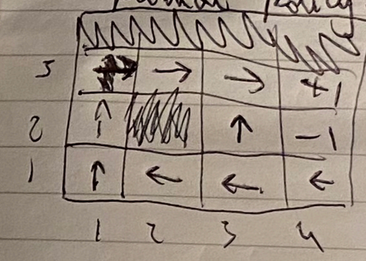
$\gamma = .999$ ← Gives smaller weight to rewards in distant future. Not "time value of money". Real reason to use it though is because it turns out using learning algos just converges faster.

Tot payoff $R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots = R_{tot}$

Goal is to choose actions over time that maximize R_{tot}.

Policy $\pi : S \mapsto A$ ← This typically comes out of using learning algos (controllers)

Optimal policy:



$\pi((3,1)) = W$