CS771

Probabilistic Regression

- To perform prob. reg. have to assign a label distribution over all $\mathbb{R}$ for every data point x using a PDF
- Suppose Gaussian distribution is used
  — need to decide μ_x, $\sigma_x^2 (>0)$
- Popular: Let $\mu_x = w^T x$, $\sigma_x^2 = \sigma^2$,
  i.e., $\mathcal{N}(\cdot \mid w^T x, \sigma^2)$ → Can choose different σ for every data point
  More complicated ←
- Likelihood func. w.r.t. (x^i, y^i) becomes:

$$\mathcal{N}(y^i \mid w^T x^i, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y^i - w^T x^i)^2 / 2\sigma^2}$$

- Log-likelihood w.r.t. a set of data points $\{(x^i, y^i)\}_{i=1}^n$

$$\min_{w \in \mathbb{R}^d} \frac{n}{2} \ln(2\pi\sigma^2) + \frac{1}{2\sigma^2} \sum_{i=1}^n (y^i - w^T x^i)^2$$

$$\min_{w \in \mathbb{R}^d} \sum_{i=1}^n (y^i - w^T x^i)^2$$

- MLE w.r.t. Gaussian likelihood indeed minimizes the least square loss
- Also note that if we set all $\sigma_x^2 = \sigma^2$, then it does not matter what σ we choose
  — will get the same model.

- Suppose Laplacian Distribution is chosen instead
  $\mu_x = w^T x$, $\sigma_x = \sigma$, i.e., $\text{Lap}(\cdot \mid w^T x, \sigma)$
- Likelihood function w.r.t. (x^i, y^i) becomes

$$\text{Lap}(y^i \mid w^T x, \sigma) = \frac{1}{2\sigma} e^{-|y^i - w^T x^i| / \sigma}$$

- Negative log likelihood w.r.t. a set of data points $\{(x^i, y^i)\}_{i=1}^n$ becomes:

$$\min_{w \in \mathbb{R}^d} n \cdot \ln(2\sigma) + \frac{1}{\sigma} \sum_{i=1}^{n} |y^i - w^T x^i| = \min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} |y^i - w^T x^i|$$

- ∴ If likelihood func. changed to use Laplacian instead, MLE ends up minimizing absolute loss.
- σ does not matter, as long as there is no regularization. σ usually treated like a hyperparameter and tuned
- Asking and accepting probabilities < 1 on the true label ~ Allowing CSVM to slack (using slack variables)

Probabilistic Regularization

- MLE often reduces to loss minimization but without regularization terms
- Can do regularization by way of priors
- L1/L2 regularization ~ specifying prob. dist.
- Prior: A P.D. over all possible models itself

Guessing the mean

- Gaussian w/ unknown mean but known variance (for simplicity) from which we receive n independent samples

$$x_1, x_2 \dots x_n \sim \mathcal{N}(\mu^*, 1)$$

- Can we estimate μ^*?
- Likelihood func for candidate μ & sample x_i

$$\mathbb{P}[x_i \mid \mu, 1] = \frac{1}{\sqrt{2\pi}} e^{\frac{-(x_i - \mu)^2}{2}}$$

- MLE: $\arg\max_{\mu \in \mathbb{R}} \prod_{i=1}^{n} \mathbb{P}[x_i \mid \mu, 1] = \arg\min_{\mu \in \mathbb{R}} \sum_{i=1}^{n} (x_i - \mu)^2$

- Suppose we believe, even before the samples, that $\mu^* \in [0, 2]$ definitely. We are said to have a "prior belief" / "prior" on model's μ. In this case, our prior is $UNIF([0,2])$. Unless we see any data to make us believe otherwise, we'll think

$$\mathbb{P}[\mu] = UNIF([0,2]) = \begin{cases} 0.5, & \mu \in [0,2] \\ 0, & \mu \notin [0,2] \end{cases}$$

- How to update μ after a prior?

Posterior

- We see data $x_1 \dots x_n$ & wish to update our belief. Want to know $\mathbb{P}[\mu \mid x_1 \dots x_n]$ (Posterior / Belief). Tells us which models are more/less likely after we have seen the data

$$\mathbb{P}[\mu \mid x_1 \dots x_n] \overset{\text{Law of total Probability}}{=} \frac{\mathbb{P}[x_1 \dots x_n \mid \mu] \cdot \mathbb{P}[\mu]}{\mathbb{P}[x_1 \dots x_n]} \overset{\text{Bayes Rule}}{=} \frac{\mathbb{P}[\mu] \cdot \prod_{i=1}^{n} \mathbb{P}[x_i \mid \mu]}{\prod_{i=1}^{n} \mathbb{P}[x_i]} \overset{\text{Independent Samples}}{=}$$

$$= \frac{\mathbb{P}[\mu] \cdot \prod_{i=1}^{n} \mathbb{P}[x_i \mid \mu]}{\prod_{i=1}^{n} \int_{\mathbb{R}} \mathbb{P}[x_i \mid t] \cdot \mathbb{P}[t]\, dt}$$

$$= \begin{cases} \dfrac{0.5 \prod_{i=1}^{n} \mathbb{P}[x_i \mid \mu]}{\prod_{i=1}^{n} 0.5 \int_0^2 \mathbb{P}[x_i \mid t]\, dt}, & \text{if } \mu \in [0,2] \\ 0, & \text{otherwise} \end{cases}$$

Maximum a Posteriori (MAP) Estimate

- Just as MLE gave us $\arg\max_{\mu \in \mathbb{R}} \mathbb{P}[x_1 \dots x_n \mid \mu, 1]$, MAP gives the model: $\arg\max_{\mu \in \mathbb{R}} \mathbb{P}[\mu \mid x_1 \dots x_n, 1] = \arg\max_{\mu \in [0,2]} \frac{0.5 \prod \mathbb{P}[x_i \mid \mu]}{\prod 0.5 \int_0^2 \mathbb{P}[x_i \mid t]\, dt}$

* Posterior prob. of some models may be larger than their prior prob. Similarly, smaller. But, 0 remains 0 ⇒ Need to be careful with our priors.

- Bad / strong priors ⟶ Funny models
- Taking negative log likelihoods:

$$\arg\min_{\mu \in \mathbb{R}} \underbrace{-\ln \mathbb{P}[\mu]}_{\text{constant for } \mu \in [0,2] \text{ \& } \infty \text{ otherwise } (\ln 0 \to -\infty)} + \tfrac{1}{2} \sum (x_i - \mu)^2 = \arg\min_{\mu \in \mathbb{R}} \sum_{i=1}^{n} (x_i - \mu)^2 \quad \text{s.t. } \mu \in [0,2]$$

- MAP can correspond to optimization problems
  → The prior became the constraint here
  → In general, the prior becomes the regularizer

- This time, instead of $\mu \in [0,2]$, we hold the that μ is not too large in magnitude $\xrightarrow{\text{same as}}$ $\mathbb{P}[\mu] = \mathcal{N}(\mu; 0, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\mu^2/2\sigma^2}$
- MAP: $\arg\min_{\mu \in \mathbb{R}} -\ln \mathbb{P}[\mu] + \tfrac{1}{2} \sum_{i=1}^{n} (x_i - \mu)^2$
  ∴ Gaussian Prior gives us L2 regularization
  $= \arg\min \frac{\mu^2}{2\sigma^2} + \tfrac{1}{2} \sum (x_i - \mu)^2 = \arg\min \frac{\mu^2}{\sigma^2} + \sum (x_i - \mu)^2$
  σ dictates the regularization constant;
  This is basically ridge regression
- Laplacian prior would give L1 regularization

Probabilistic Regularization

- $\mathbb{P}[y^i \mid x^i, w] = \mathcal{N}(y^i \mid w^T x^i, \sigma_i^2) = \frac{e^{\frac{-(y^i - w^T x^i)^2}{2\sigma_i^2}}}{\sqrt{2\pi\sigma_i^2}}$

This is the same as saying $y^i = w^T x^i + \epsilon^i$ where $\epsilon^i \sim \mathcal{N}(0, \sigma^2)$

- Let us assume we want small L2 norm, i.e., our prior is $\mathcal{N}(0, I_d)$: $\mathbb{P}[w] = \mathcal{N}(w \mid 0, \sigma_p^2 \cdot I_d) = \frac{1}{(2\pi\sigma_p^2)^{d/2}} e^{-\|w\|_2^2 / 2\sigma_p^2}$
- Posterior: $\mathbb{P}[w \mid y^1 \dots y^n, x^1 \dots x^n] = \frac{\mathbb{P}[y^1 \dots y^n \mid x^1 \dots x^n w] \cdot \mathbb{P}[w]}{\mathbb{P}[y^1 \dots y^n x^1 \dots x^n]}$
$= \frac{\mathbb{P}[w] \cdot \prod_{i=1}^n \mathbb{P}[y^i \mid x^i, w]}{\prod_{i=1}^n \mathbb{P}[y^i, x^i]}$

Ignoring terms that don't involve w, taking logs gives us $\hat{w}_{MAP} = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} -\ln \mathbb{P}[w] - \sum_{i=1}^{n} \ln \mathbb{P}[y^i \mid x^i, w]$

* Likelihood distribution over labels ($\mathbb{R}$) while prior & posterior are dist. over models ($\mathbb{R}^d$)

- For Gaussian likelihood $\mathcal{N}(y^i \mid w^T x^i, \sigma_\ell)$ and Gaussian prior $\mathcal{N}(w \mid 0, \sigma_p^2 \cdot I_d)$, we get $\hat{w}_{MAP} = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \frac{\|w\|_2^2}{\sigma_p^2} + \frac{1}{\sigma_\ell^2} \sum_{i=1}^{n} (y^i - w^T x^i)^2$

Ridge regression: $\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \left(\frac{\sigma_\ell}{\sigma_p}\right)^2 \|w\|_2^2 + \sum_{i=1}^{n} (y^i - w^T x^i)^2$

↳ σ_ℓ, σ_p together decide regularization constant

- ∃ Multivariate version of Laplace dist. too
Using it as a prior w/ $\mu = 0$ gives us LASSO

$$\mathcal{L}(w \mid \mu, \sigma) = \frac{1}{(2\sigma)^d} e^{-\frac{\|w - \mu\|_1}{\sigma}}$$

- If we think w is near a vector v, we can use $\mathcal{N}(w \mid v, \sigma_p^2)$ instead

Bayesian Learning

- Instead of trusting a single model, we place partial trust, possibly over all models
→ Models w/ higher posterior prob. value get higher trust.

· Use Bayes rule for these calculations

From PML to BML

· We have (x^i, y^i) $\mathbb{P}[w]$ (prior over models)
For test point x_t, want to output a dist. over set of all labels, i.e, $\mathbb{P}[y \mid x^t, \{x^i, y^i\}]$

· We now introduce models:

$$\mathbb{P}[y \mid x^t, \{x^i, y^i\}] = \int_{\mathbb{R}^d} \mathbb{P}[y, w \mid x^t, \{x^i, y^i\}]\, dw$$

$$= \int_{\mathbb{R}^d} \mathbb{P}[y \mid w, x^t, \{x^i, y^i\}] \cdot \mathbb{P}[w \mid x^t, \{x^i, y^i\}]\, dw$$

$$= \int_{\mathbb{R}^d} \underbrace{\mathbb{P}[y \mid w, x^t]}_{\text{Prediction of } w \text{ on } x^t} \cdot \underbrace{\mathbb{P}[w \mid \{x^i, y^i\}]}_{\text{Faith in } w} dw$$

↓ Predictive Posterior

Bayesian Regression

· Suppose we have Gaussian likelihood $\mathcal{N}(y^i \mid w^T x^i, \sigma_l^2)$ and Gaussian prior $\mathcal{N}(w \mid 0, \sigma_p^2 \cdot I_d)$. Then we have $\mathbb{P}[w \mid \{x^i, y^i\}] = \mathcal{N}(w; \hat{\mu}, \hat{\Sigma})$,

$$\hat{\mu} = (x^T x + (\sigma_l/\sigma_p)^2 \cdot I_d)^{-1} \cdot x^T y$$ — and

$$\hat{\Sigma} = \frac{1}{\sigma_p^2} (x^T x + (\sigma_l/\sigma_p)^2 \cdot I_d)^{-1}$$ → dimply the MAP solution

Makes sense ∵ MAP : mode of posterior &
for Gaussian dist, mode = mean

* Predictive Posterior: $\mathbb{P}[y \mid x, \{x^i, y^i\}] = \mathcal{N}(y; \hat{\mu}_x, \hat{\sigma}_x^2)$
where $\hat{\mu}_x = \hat{\mu}^T x$ and $\hat{\sigma}_x^2 = \sigma_l^2 + x^T \hat{\Sigma} x$
→ Variance of predicted dist. $\hat{\sigma}_x^2$ depends on the point itself.

· Some likelihood-prior pairs are special:

- Always yield a posterior of the same family as the prior
- Such pairs are called conjugate pairs
- "Prior" said to be the conjugate to "Likelihood"
- Ex: Gaussian-Gaussian pair

Generative ML

- The data features were always considered constant and never questioned to be random or flexible
  - → Can we talk about $\mathbb{P}[x \mid y, \theta]$?
  - → Given y, can generate a new x

Generative Algorithms

- Algos that can learn dist. of the form $\mathbb{P}[x \mid y]$, $\mathbb{P}[x, y]$, $\mathbb{P}[x]$
- Discriminative algorithms: only use $\mathbb{P}[y \mid x]$
  Generative algorithms: Use $\mathbb{P}[x \mid y]$, $\mathbb{P}[x, y]$, $\mathbb{P}[x]$, etc.

Advantages
- More expensive: Slower train, test times; Larger models
- Overkill: Often need only $\mathbb{P}[y \mid x]$ for predictions

Disadvantage
- More frugal: Can work even if less training data
- More robust: Can work even if features corrupted / missing

A simple Generative Model

- Given a few feature vectors $x^1 \ldots x^n \in \mathbb{R}^d$
- Wish to learn a P.D. $\mathbb{P}[\cdot]$ w/ support over $\mathbb{R}^d$
- Let's try to learn a Gaussian dist., i.e., $\mu \in \mathbb{R}^d$ so that $\mathcal{N}(\mu, I_d)$ explains the data well

- One way is to look for μ maximizing likelihood
- As before, assume independence in x^i

$$\arg\max_{\mu \in \mathbb{R}^d} \mathbb{P}[x^1 \dots x^n \mid \mu, I_d] = \arg\min_{\mu \in \mathbb{R}^d} \sum_{i=1}^{n} \|x^i - \mu\|_2^2$$

↳ FOC ⟶ $\hat{\mu}_{MLE} = \sum_{i=1}^{n} x^i / n$

- We learnt $\mathcal{N}(\hat{\mu}_{MLE}, I_d)$

- Now, we want to learn μ as well as $\sigma \geq 0$ so that $\mathcal{N}(\mu, \sigma^2 \cdot I_d)$ explains the data well

- Log-likelihood: $\arg\max_{\substack{\mu \in \mathbb{R}^d \\ \sigma \geq 0}} \ln \mathbb{P}[x^1 \dots x^n \mid \mu, \sigma^2] = \arg\min_{\mu, \sigma} dn \ln \sigma + \sum \frac{\|x^i - \mu\|_2^2}{2\sigma^2}$

FOC w.r.t. μ : $\hat{\mu}_{MLE} = \sum x^i / n$

FOC w.r.t. σ : $\hat{\sigma}^2_{MLE} = \sum \|x^i - \hat{\mu}_{MLE}\|_2^2 / d \cdot n \geq 0$

↳ Global optima

- Now, we want to learn μ & $\Sigma \geq 0$ {Σ: PSD} for dist. $\mathcal{N}(\mu, \Sigma)$

$\Rightarrow \arg\max_{\mu, \Sigma} \ln \mathbb{P}[x^1 \dots x^n \mid \mu, \Sigma] = \arg\min_{\mu, \Sigma} f(\mu, \Sigma)$

$$= \arg\min_{\mu, \Sigma} \frac{n}{2} \ln |\Sigma| + \frac{1}{2} \sum_i (x^i - \mu)^T \Sigma^{-1} (x^i - \mu)$$

FOC w.r.t. μ : $(\Sigma^{-1} + (\Sigma^{-1})^T) \sum_{i=1}^{n} (x^i - \mu) = 0$

↳ Holds for $\mu = \hat{\mu}_{MLE}$ but can hold for other cases too $\Rightarrow$ multiple optima

{$\Sigma^{-1} + (\Sigma^{-1})^T$ not invertible}

- For $X \in \mathbb{R}^{d \times d}$, $tr(X) := \sum_{i=1}^{d} X_{ii}$

* If $a \in \mathbb{R}^d$, then $a^T X a = tr(A^T X)$, $A = a a^T \in \mathbb{R}^{d \times d}$

If A constant matrix, then $\frac{\partial\, tr(A^T X)}{\partial X} = A$

- Denote $\Lambda = \Sigma^{-1}$, then

$$f(\mu, \Sigma) = n/2 \ln|\Sigma| + \sum_{i=1}^{n} tr(\Lambda^T S^i)/2,$$
$$S^i = (x^i - \mu)(x^i - \mu)^T$$

- For any $A, B, C \in \mathbb{R}^{d \times d}$,

$tr(A^TB) = tr(B^TA)$; $tr(A^TB) + tr(A^TC) = tr(A^T(B+C))$

$\Rightarrow f(\mu, \Sigma) = n/2 \ln|\Sigma| + tr(\Lambda^T S)/2$, $S = \sum_{i=1}^{n} S_i$

- $\frac{\partial \ln|\Sigma|}{\partial \Sigma} = \frac{1}{|\Sigma|} \frac{\partial |\Sigma|}{\partial \Sigma} = \frac{|\Sigma| \cdot (\Sigma^{-1})^T}{|\Sigma|} = (\Sigma^{-1})^T = \Sigma^{-1}$ (assume symmetry)

$$\frac{\partial\, tr(S^T\Lambda)}{\partial \Sigma} = \frac{\partial \Lambda}{\partial \Sigma} \cdot \frac{\partial\, tr(S^T\Lambda)}{\partial \Lambda} = -\Sigma^{-2} S$$

- $\therefore$ FOC w.r.t. Σ : $n/2\, \Sigma^{-1} - 1/2\, \Sigma^{-2} S = 0 0^T$

$\Rightarrow \hat{\Sigma}_{MLE} = S/n = \sum_{i=1}^{n} (x^i - \mu)(x^i - \mu)^T / n$

$\because \hat{\Sigma}_{MLE} \succcurlyeq 0$ & symmetric, it must be the global optimum

- Now, suppose we are concerned that one Gaussian cannot capture all the variations
  - ↳ can we learn 2 (or more) Gaussians?
    - ↳ Mixture of Gaussians

Expectation Maximization (EM) Algorithm used

→ We suspect that instead of one, two Gaussians are involved → Call them Each is a component ← $\mathcal{N}(\mu^1, I_d)$, $\mathcal{N}(\mu^2, I_d)$

$\because$ We are unsure which data point came from where, we introduce a **latent variable**, $z_i \in \{1, 2\}$ per data point
  - ↳ Hidden / dormant / Concealed

→ If someone tells us that $z_i = 1$ then this means that the 1st Gaussian is responsible for that data point and the likelihood expression is:

$$\mathbb{P}[x^i \mid z_i = 1, \mu^1, \mu^2] = \mathcal{N}(x^i; \mu^1)$$

Similarly for $z_i = 2$.

Latent variables can be discrete / continuous

- Wish to obtain the maximum (log) likelihood models, i.e., $\underset{\mu^1, \mu^2 \in \mathbb{R}^d}{\operatorname{argmax}} \sum_{i=1}^n \ln \mathbb{P}[x^i \mid \mu^1, \mu^2]$

∵ we don't know the values of z_i, force them into the exp. using the law of total prob.

$$\underset{\mu^1, \mu^2 \in \mathbb{R}^d}{\operatorname{argmax}} \sum_{i=1}^n \ln \left(\sum_{c \in \{1,2\}} \mathbb{P}[x^i, z_i = c \mid \mu^1, \mu^2] \right)$$

Assume that $\mathbb{P}[z^i \mid \mu^1, \mu^2] = \text{const.}$ and that only one of the terms in Σ_c will dominate → Convert to a double maximization problem

$\Rightarrow \Sigma_c \approx \max_c$

$$\underset{\mu^1, \mu^2}{\operatorname{argmax}} \underset{z_i}{\operatorname{argmax}} \sum_{i=1}^n \ln \mathbb{P}[x^i \mid z_i, \mu^1, \mu^2]$$

→ No longer $\Sigma \ln(\Sigma)$. Looks like an MLE problem

→ Still difficult, but the following is easy:

H1

S1: Fix $\mu^1, \mu^2 \in \mathbb{R}^d$ and update latent vars. z_i to their optimal values $\underset{z_i}{\operatorname{argmax}} \sum_{i=1}^n \ln \mathbb{P}[x^i \mid z_i, \mu^1, \mu^2]$

S2: Fix z_i & update $\mu^1, \mu^2 \in \mathbb{R}^d$ to optimal values $= \underset{\mu_1, \mu_2}{\operatorname{argmax}} \sum_{i=1}^n \ln \mathbb{P}[x^i \mid z_i, \mu^1, \mu^2]$

- Keep alternating b/w S1 & S2

S1 ⇝ $\underset{z_i \in \{1,2\}}{\operatorname{argmax}} \ln \mathbb{P}[x^i \mid z_i, \mu^1, \mu^2] = \underset{z_i \in \{1,2\}}{\operatorname{argmin}} \| x^i - \mu^{z_i} \|_2^2$

↕

S2 ⇝ $\underset{\mu^1, \mu^2}{\operatorname{argmax}} \sum_{i=1}^n \ln \mathbb{P}[x^i \mid z_i, \mu^1, \mu^2]$

$= \underset{\mu^1 \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{\substack{i: \\ z_i = 1}} \| x^i - \mu^1 \|_2^2 + \underset{\mu^2}{\operatorname{argmin}} \sum_{\substack{i: \\ z_i = 2}} \| x^i - \mu^2 \|_2^2$

Repeat

≡ k means Algorithm

$\Rightarrow \mu_1 = \frac{1}{n_1} \sum_{i}^{z_i = 1} x^i \; ; \; \mu_2 = \frac{1}{n_2} \sum_{i}^{z_i = 2} x^i$,

$n_c :=$ # of data points w/ $z_i = c$

H2

S1 (E Step)

S1.1: Assume current estimates are $\mu^1 = p, \mu^2 = q$

Use current models to ascertain how likely z_i values are for i^{th} data point, compute

$q_c^i = \mathbb{P}[z_i = c \mid x^i, \mu^1, \mu^2]$ for both $c \in \{1, 2\}$

S1.2: Use weights q_c^i to set up a new objective func. (assume $\mathbb{P}[z_i \mid \mu^1, \mu^2] = \text{const.}$)

$$\sum_{i=1}^{n} \sum_{c \in \{1,2\}} q_c^i \cdot \ln \mathbb{P}[x^i \mid z_i = c, \mu^1, \mu^2]$$

S2 (M Step): Maximize new obj func. $\operatorname{argmax}_{\mu_1, \mu_2 \in \mathbb{R}^d}(\;)$

H1: Heuristic 1 - Alternating Optimization

H2: Heuristic 2 - Expectation Maximization

Derivation of E Step:

θ: Denotes the models μ^1, μ^2; θ^0: Current estimate

$$\ln \mathbb{P}[x^i \mid \theta] = \ln \left(\sum_{c \in \{1,2\}} \mathbb{P}[x^i, z_i = c \mid \theta] \right)$$

$$= \ln \left(\sum_c \mathbb{P}[z_i = c \mid x^i, \theta] \cdot \frac{\mathbb{P}[x^i, z_i = c \mid \theta]}{\mathbb{P}[z_i = c \mid x^i, \theta^0]} \right)$$

Jensen's Inequality ↑

$$\geq \sum_c \mathbb{P}[z_i = c \mid x^i, \theta] \cdot \ln \left(\mathbb{P}[x^i, z_i = c \mid \theta] / \mathbb{P}[z_i = c \mid x^i, \theta^0] \right)$$

$$= \sum_c q_c^i \cdot \ln \left(\mathbb{P}[x^i, z_i = c \mid \theta] / q_c^i \right)$$

e^i doesn't depend on θ (θ^0 instead) →

$$= \sum_c q_c^i \ln \mathbb{P}[x^i, z_i = c \mid \theta] + e_i$$

EM for GMM

Gaussian Mixture Models

1. Init means $\{\mu^c\}_{c=1,\dots,C}$
2. update π_c^i using $\{\mu^c\}$ $\forall\, i \in [n]$

$$p_c^i \leftarrow e^{-\|x^i - \mu^c\|_2^2/2}, \quad q_c^i \leftarrow p_c^i / \sum_{c=1}^{C} p_c^i$$

3. $n_c = \sum_{i=1}^{n} q_c^i$
4. $\mu^c \leftarrow 1/n_c \sum_{i=1}^{n} q_c^i \cdot x^i$
5. Repeat

Con: EM usually more expensive to execute than alternating optimization

Pro: EM ensures that obj. value of original problem, i.e.,

$$\sum_{i=1}^{n} \ln \left(\sum_{c \in \{1,2\}} \mathbb{P}[x^i, z_i = c \mid \mu^1, \mu^2] \right)$$

always ↑ → No guarantee about global maximum

May converge & get stuck at a local max. instead

Q Function

- Let $Q_t(\theta) = \sum_{i=1}^{n} \sum_{c \in \{1,2,3\}} q_c^{i,t} \ln \mathbb{P}[x^i, z_i = c \mid \theta] + e_i$ be the new obj. func. at time t
- EM algo constructs a new Q_t at each time during the E-step and maximizes it during the M-step $\theta^{t+1} = \text{argmax}\ Q_t(\theta)$
- We have already seen $Q_t(\theta) \leq \sum_{i=1}^{n} \ln \mathbb{P}[x^i \mid \theta]\ \forall \theta$
  Also, $Q_t(\theta^t) = \sum_{i=1}^{n} \ln \mathbb{P}[x^i \mid \theta^t]$
  Some indication as to why EM ↑ likelihood each iteration
- Alternating optimization can be seen as a cousin of EM that uses a Q func. of the form
  $Q_t(\theta) = \sum_{i}^{n} \ln \mathbb{P}[x^i, \tilde{z}_i \mid \theta]$, $\tilde{z}_i = \underset{z_i}{\text{argmax}} \ln \mathbb{P}[x^i \mid z_i, \theta^t]$

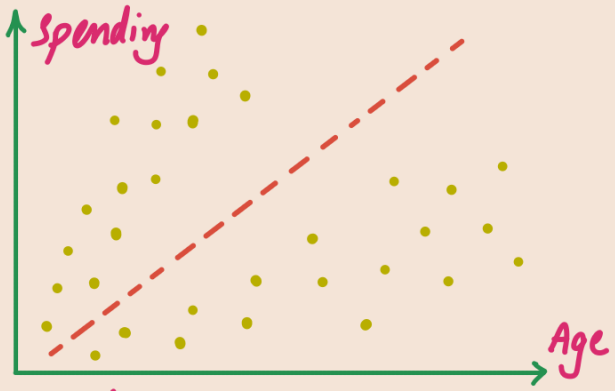
Generic EM Algo

1. Init model θ^0
2. $\forall\ i\ \forall\ z \in \mathcal{Z} : q_z^{i,t} \leftarrow \mathbb{P}[z_i = z \mid x^i, \theta^t]$
3. $Q_t(\theta) = \sum_{i=1}^{n} \sum_{z \in \mathcal{Z}} q_z^{i,t} \ln \mathbb{P}[x^i, z^i = z \mid \theta]$
4. $\theta^{t+1} = \text{argmax}\ Q_t(\theta)$
5. Repeat

* Might get stuck at a local extrema in $Q_t(\cdot)$ funcs.

Mixed Regression

Ex: Age v/s Spending



- Two subpopulations in data (gender) which behave differently even if age is same
- Could try clustering first but data spread around a line and not a centroid, so latent vars. better
  We work with a mixed model (aka mixture of experts). Will fit 2 regression models and use a

Latent variable $z_i \in \{1,2\}$ to keep track.

Let's use Gaussian likelihoods

$\mathbb{P}[y^i | x^i, z_i = 1, w^1, w^2] = \mathcal{N}(y^i; \langle w^1, x^i \rangle, \sigma^2)$

$\mathbb{P}[y^i | x^i, z_i = 2, w^1, w^2] = \mathcal{N}(y^i; \langle w^2, x^i \rangle, \sigma^2)$

- Could have had separate σ_1^2 and σ_2^2
  More tedious $\Rightarrow$ Assume $\sigma_1^2 = \sigma_2^2 = \sigma^2 = 1$

Note: This is not generative learning $\because$ we are still learning discriminative distributions $\mathbb{P}[y^i | x^i, w^c]$

MLE for Mixed Regression

$$\underset{w^1, w^2 \in \mathbb{R}^d}{\operatorname{argmax}} \sum_{i=1}^n \ln \mathbb{P}[y^i | x^i, w^1, w^2] \xrightarrow{\text{Introducing } z_i} \underset{w^1, w^2 \in \mathbb{R}^d}{\operatorname{argmax}} \sum_{i=1}^n \ln \left(\sum_{c \in \{1,2\}} \mathbb{P}[y^i, z_i = c | x^i, w^1, w^2] \right)$$

M1 Alternating Maximization

Assume $\mathbb{P}[z^i | x^i, w^1, w^2] = \text{const}$.

$\mathbb{P}[y^i, z_i = c | x^i, w^1, w^2] \rightsquigarrow \mathbb{P}[y^i | z_i = c, x^i, w^1, w^2]$

$\mathbb{P}[A, B | C] = \mathbb{P}[A \cap B \cap C] / \mathbb{P}[C] = \mathbb{P}[A \cap B \cap C] / \mathbb{P}[B \cap C] \cdot \mathbb{P}[B \cap C] / \mathbb{P}[C] = \mathbb{P}[A | B, C] \cdot \mathbb{P}[B | C]$

S1: Fix w^1, w^2 & $z_i \leftarrow \underset{c \in \{1,2\}}{\operatorname{argmax}} \ln \mathbb{P}[y^i | x^i, z_i = c, w^1, w^2]$

S2: Fix all z_i & $(w^1, w^2) \leftarrow \underset{w^1, w^2 \in \mathbb{R}^d}{\operatorname{argmax}} \sum_{i=1}^n \ln \mathbb{P}[y^i | x^i, z_i, w^1, w^2]$

S1: $\underset{c \in \{1,2\}}{\operatorname{argmax}} \ln \mathbb{P}[y^i | x^i, z_i = c, w^1, w^2] = \underset{c \in \{1,2\}}{\operatorname{argmin}} |y^i - \langle w^c, x^i \rangle|$

{Assign each point to its closest line}

S2: $\underset{w^1, w^2 \in \mathbb{R}^d}{\operatorname{argmax}} \sum_{i=1}^n \ln \mathbb{P}[y^i | x^i, z_i, w^1, w^2]$

$= \underset{w^1}{\operatorname{argmin}} \sum_{i : z_i = 1} (y^i - \langle w^1, x^i \rangle)^2 + \underset{w^2}{\operatorname{argmin}} \sum_{z_i = 2} (y^i - \langle w^2, x^i \rangle)^2$

{Perform least squares on data points assigned to each component}

Alt Opt for MR

1. Init models $\{w^c\}_{c \in \{1,2\}}$
2. $z_i \leftarrow \underset{c}{\operatorname{argmin}} |y^i - \langle w^c, x^i \rangle| \quad \forall\, i \in [n]$
3. $w^c \leftarrow \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i : z_i = c} (y^i - \langle w^c, x^i \rangle)^2$
4. Repeat

M2 EM for Mixed Regression

E step

S1.1: Assume $w^1 = p$, $w^2 = q$. Use these to ascertain likelihoods of value of z_i: $q_c^i \leftarrow \mathbb{P}[z_i = c \mid y^i, x^i, p, q]$ $\forall\, c \in \{1,2\}$

S1.2: Assume $\mathbb{P}[z_i \mid x^i, p, q] = \text{const.}$

Use q_c^i to setup new obj. func. $\sum_{i=1}^{n} \sum_{c \in \{1,2\}} q_c^i \cdot \ln \mathbb{P}[y^i \mid x^i, z_i = c, w^1, w^2]$

M step

S2: Maximize new objective function: $\operatorname{argmax}_{w^1, w^2 \in \mathbb{R}^d} \sum_{i=1}^{n} \sum_{c \in \{1,2\}} q_c^i \cdot \ln \mathbb{P}[y^i \mid x^i, z_i = c, w^1, w^2]$ (Repeat)

EM for MR

1. Init models $\{w^c\}_{c=1 \dots C}$
2. $p_c^i \leftarrow e^{-\|y^i - \langle w^c, x^i \rangle\|_2^2/2}$; $q_c^i \leftarrow p_c^i / \sum_{c=1}^{C} p_c^i$ $\forall\, i \in [n]$
3. $w^c \leftarrow \operatorname{argmin}_{w \in \mathbb{R}^d} \sum_{i=1}^{n} q_c^i \cdot \|y^i - \langle w^c, x^i \rangle\|_2^2$
   $= \operatorname{argmin} (Xw - y)^T Q_c (Xw - y) = (X^T Q_c X)^{-1} (X^T Q_c y)$

   where $Q_c = \operatorname{diag}(q_c^1 \dots q_c^n)^W$
4. Repeat

Generative Supervised Learning

"For each class $c \in [C]$ learn what do data points of that class look like (using a dist.). For a test data point, ask each of these C dists. to vote based on how much they think the data point belongs to their class."

Interpreting the above in lang. of prob, for a test x^t, predict the label $\hat{y}^t$ based on the following rule (θ is the model)

$$\hat{y}^t = \operatorname*{argmax}_{c \in [C]} \mathbb{P}[y^t = c \mid x^t, \theta] \quad \text{(Bayes' Rule)}$$
$$= \operatorname*{argmax}_{c \in [C]} \mathbb{P}[x^t \mid y^t = c, \theta] \cdot \mathbb{P}[y^t = c \mid \theta]$$

Ex: $C = 2$, $\mathbb{P}[y^t = 1 \mid \theta] = p \leftrightarrow \mathbb{P}[y^t = -1 \mid \theta] = 1 - p$ (Rademacher)

prior ← $\mathbb{P}[y^t = 1 \mid x^t, \theta]$ → posterior

· If multiclass, would have used multinolli.

· For $P[x^t | y^t = c, \theta]$ ⟵ generative model for feature vectors of class c aka class conditional dist.

Let's choose a ⟵ single Gaussian

$$P[x^t | y^t = 1, \theta] = \mathcal{N}(\mu^+, \Sigma^+) \quad \& \quad P[x^t | y^t = -1, \theta] = \mathcal{N}(\mu^-, \Sigma^-) \quad \text{with}$$

$$\theta = \{\{\mu^+, \Sigma^+\}, \{\mu^-, \Sigma^-\}, p\}$$

All that is left is to <u>estimate</u> these parameters

Will use MLE; can have priors

∵ Both x and y are getting modelled here, likelihood function now looks at the joint prob. of $P[x, y | \theta]$

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}}\, P[x^1 y^1 \dots x^n y^n | \theta] \quad \text{(Independence)}$$

$$= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} P[x^i, y^i | \theta] \quad \text{(Chain Rule)}$$

$$= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} P[x^i | y^i, \theta] \cdot P[y^i | \theta]$$

If $y^i = +1$, then μ^-, Σ^- not relevant

$$= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} P[x^i | y^i, \mu^{y^i}, \Sigma^{y^i}] \cdot P[y^i | p] \quad \text{(Take logs)}$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \ln P[x^i | y^i, \mu^{y^i}, \Sigma^{y^i}] + \ln P[y^i | p]$$

Breaks down into

$$\hat{p}_{MLE} = \underset{p}{\operatorname{argmax}} \sum_{i=1}^{n} \ln P[y^i | p]$$

$$\{\hat{\mu}^+_{MLE}, \hat{\Sigma}^+_{MLE}\} = \underset{\mu^+, \Sigma^+}{\operatorname{argmax}} \sum_{i:y^i=1} \ln P[x^i | y^i, \mu^+, \Sigma^+]$$

$$\{\hat{\mu}^-_{MLE}, \hat{\Sigma}^-_{MLE}\} = \underset{\mu^-, \Sigma^-}{\operatorname{argmax}} \sum_{i:y^i=-1} \ln P[x^i | y^i, \mu^-, \Sigma^-]$$

$$\hat{p}_{MLE} = \underset{p}{\operatorname{argmax}}\, n_+ \cdot \ln p + n_- \cdot \ln(1-p)$$

$$\hat{p}_{MLE} = \frac{n_+}{n_+ + n_-} = \frac{n_+}{n}$$

n_+ : No. of data points with label 1. Similarly, n_-

$$\hat{\mu}^+_{MLE} = (1/n_+) \sum_{i:y^i=1} x^i, \quad \hat{\Sigma}^+_{MLE} = (1/n_+) \sum_{i:y^i=1} (x^i - \hat{\mu}^+_{MLE})(x^i - \hat{\mu}^+_{MLE})^T$$

Similarly for $\hat{\mu}^-_{MLE}$ and $\hat{\Sigma}^-_{MLE}$

Special Cases

I) Fix $\Sigma^+ = \Sigma^- = I_d$

Predict $\hat{y}^t = 1$ only if $e^{-\|x^t - \hat{\mu}^+\|_2^2/2} \cdot \frac{n^+}{n} \geq e^{-\|x^t - \hat{\mu}^-\|_2^2/2} \cdot \frac{n^-}{n}$

↕ Linear Classifier

$w = 2(\hat{\mu}^+ - \hat{\mu}^-)$, $b = \|\hat{\mu}^-\|_2^2 - \|\hat{\mu}^+\|_2^2 + 2\ln(n^+/n^-)$: $w^T x + b \geq 0$

↳ Standard Gaussian has independent coordinates

⇒ Implicit assumption that diff. coords. of x^t are ind.

⇒ Naïve Bayes model

II) $\Sigma^+ = \Sigma^- = \Sigma$ (not necessarily I_d)

$\hat{\mu}^+_{MLE} = \frac{\sum_{y^i=1} x^i}{n^+}$, $\hat{\mu}^-_{MLE} = \frac{\sum_{y^i=-1} x^i}{n^-}$, $\hat{\Sigma}_{MLE} = \frac{1}{n}\sum_{i=1}^{n} (x^i - \hat{\mu}^{y^i}_{MLE})(x^i - \hat{\mu}^{y^i}_{MLE})^T$

Decision boundary: remains linear

- Often called "Linear Discriminant Analysis" or "Fisher's linear discriminant"

III) $p = 0.5$; $\Sigma^+ = \Sigma^- = I_d$

Predict $\hat{y}^t = 1$ only if $\|x^t - \hat{\mu}^+\|_2^2 \leq \|x^t - \hat{\mu}^-\|_2^2$

↳ Exactly LwP

- If we fix to Σ instead of I_d, we still get LwP but with a "Mahalanobis" distance instead

General Case: Each class gets its own separate Gaussian, with the decision boundary as a quadratic function.

- Let $\hat{\Lambda}^+ := (\hat{\Sigma}^+)^{-1}$, $\hat{\Lambda}^- := (\hat{\Sigma}^-)^{-1}$

$x^T A x + x^T b + c \geq 0$, where $A = \hat{\Lambda}^- - \hat{\Lambda}^+$, $b = 2(\hat{\Lambda}^+\hat{\mu}^+ - \hat{\Lambda}^-\hat{\mu}^-)$,

and $c = (\hat{\mu}^-)^T\hat{\Lambda}^-\hat{\mu}^- - (\hat{\mu}^+)^T\hat{\Lambda}^+\hat{\mu}^+ + 2\ln\left(\frac{n_+}{n_-}\right) + \ln\frac{|\hat{\Lambda}^+|}{|\hat{\Lambda}^-|} \geq 0$

Missing Data

- Assume for simplicity that only last k coordinates go missing, i.e., $x^t = [x_o^t, x_m^t] \in \mathbb{R}^d$, $x_o^t \in \mathbb{R}^{d-k}$ (observed), $x_m^t \in \mathbb{R}^k$ (missing)
- In such cases, Generative Learning can help reconstruct the data point as well as classify it correctly — use marginal probability

$$P[x_o^t \mid y^t = c, \theta] = \int_{\mathbb{R}^k} P[[x_o^t, v] \mid y^t = c, \theta]\, dv$$

- All marginals of Gaussian are Gaussian:
  - → If $x \sim N(\mu, \Sigma)$ then $x_o \sim N(\mu_o, \Sigma_{oo})$
  - → can use this to classify x_o into some $\hat{y}^t$
  - → Then using the predicted $\hat{y}^t$, we can fill in missing features → called "Feature Imputation"

$$\hat{x}_m^t = \underset{v}{\text{argmax}}\, P[v \mid x_o^t, \hat{y}^t, \theta] = \hat{\mu}_m^{y^t} + \Sigma_{mo}^{\hat{y}^t} (\Sigma_{oo}^{\hat{y}^t})^{-1} (x_o^t - \mu_o^{\hat{y}^t})$$

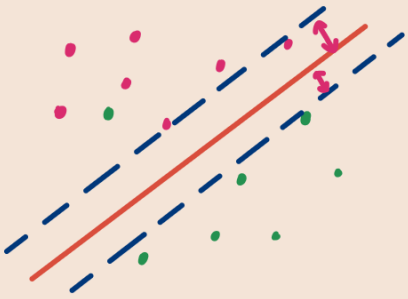
- Conditionals of Gaussian are Gaussian too

▷ What if feature not missing but supposed to be 0?
▷ What if training data incomplete, eg. missing features or completely wrong labels?
→ Require more sophisticated methods like Adversarial learning, Robust learning, etc.

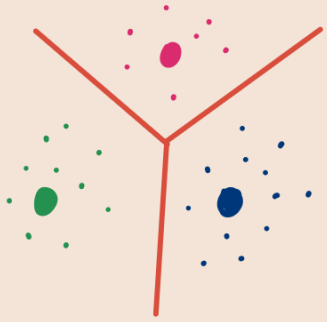
Learning with Kernels

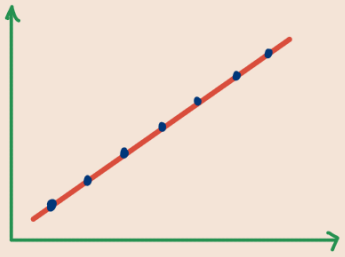
Kernel methods are a good option to try whenever linear models do a poor job on our data

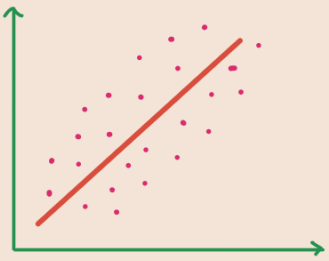
Use cases of a Linear Classifier (LC)



Fits w/ or w/o a few outliers

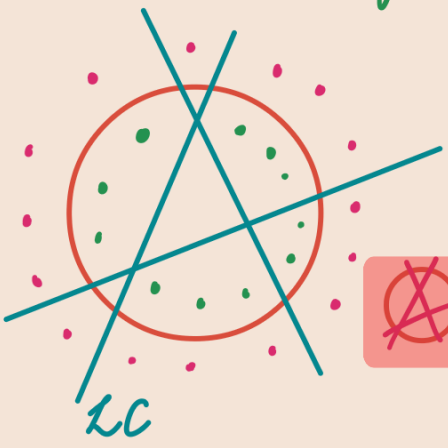




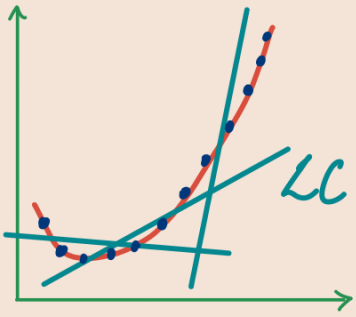


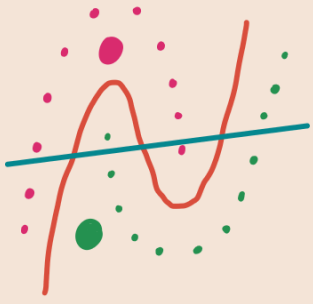
Fits well, even w/ some noise

Linear Classifiers do not do well (NLC, LC)



LC leads to ~~Anarchy~~





Blessing of Dimensionality

Collectively called the Curse of dimensionality

- Dimensionality ↑ ⇒ Hardships ↑ ← Criticism
  - Algorithms become expensive
  - much more training data required
- → Allows powerful and flexible models

Case: Consider n feature vectors in d ($\geq n$) dimensions with all n feature vectors linearly independent, i.e., $\text{rank}(X) = n$

→ Perfect regression is possible for this data:

Let $y \in \{-1, 1\}^n$ label vector for classification, $y \in \mathbb{R}^n$ for reg.

$X = U \Sigma V^T$ (thin) SVD of feature matrix X; $U, \Sigma \in \mathbb{R}^{n \times n}$, $V \in \mathbb{R}^{d \times n}$, $V^T V = I_d = U U^T = U^T U$

Curse of Dimensionality
A collection of unintuitive

things that happen in layd dimensions

Trust the math

$VV^T \neq I_n$ possible $\because d \geq n$. Σ invertible $\because$ thin SVD

Then $w = V\Sigma^{-1}U^T y$

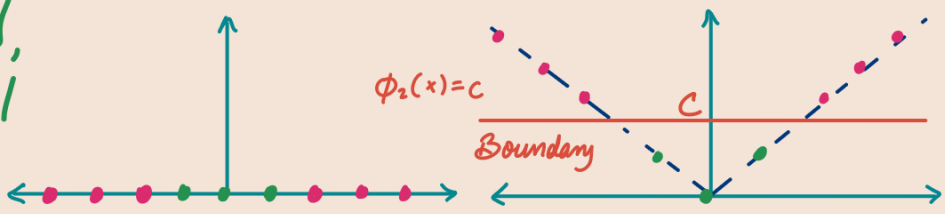
satisfies $Xw = y$ $\{Xw = (U\Sigma V^T)(V\Sigma^{-1}U^T y)\}$

Non-linear Classification

Ex: $\mathbb{R} \ni x \mapsto \phi(x) = [x, |x|] \in \mathbb{R}^2$

$\phi(x) = [\phi_1(x), \phi_2(x)]$,

$\phi_1(x) = x$, $\phi_2(x) = |x|$



- Original data non-separable by any linear classifier but using a nice non-linear map makes them separable w/ margin
- Decision boundary is non-linear in original space $\mathbb{R}$, i.e., linear model in $\mathbb{R}^2$ imposed a non-linear boundary in $\mathbb{R}$ (non-linearity came from map ϕ)

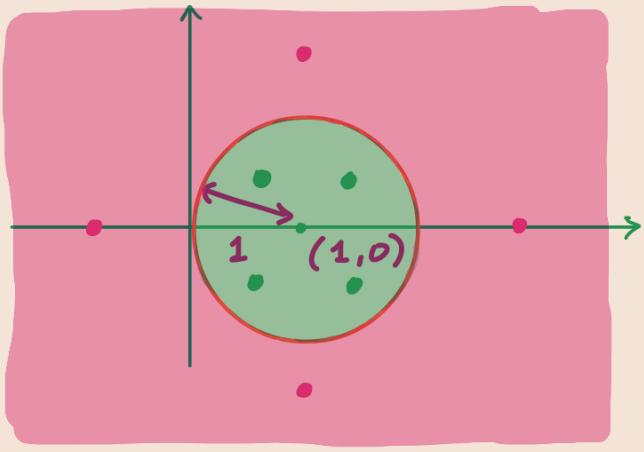
* $\nexists\ w \in \mathbb{R}^{2\times 1}$: $\phi(x) = wx$ $\forall\ x \in \mathbb{R}$

Ex: $\mathbb{R}^2 \ni (x, y)$

$\mapsto \phi(x,y) = (x, y^2, x^2) \in \mathbb{R}^2$

A linear func. on the mapped feature vectors looks like

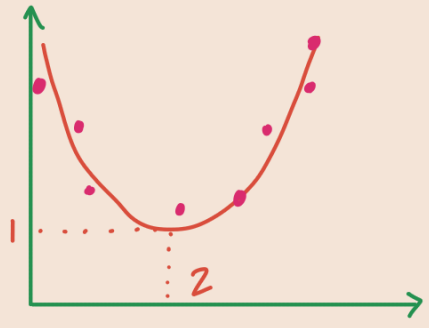
$\langle w, \phi(x,y) \rangle = w_1 x + w_2 y^2 + w_3 x^2$



Consider $w = (-2, 1, 1)$, corresponds to $-2x + y^2 + x^2 = 0$

$\equiv (x-1)^2 + y^2 = 1$

Ex:



$\mathbb{R} \ni x \mapsto \phi(x) = [1, x, x^2] \in \mathbb{R}^3$

$y = (x-2)^2 + 1$ fits closely & can be easily written as

$\langle w, \phi(x) \rangle$, $w = \langle 1, -4, 5 \rangle$, $\phi(x) = \langle x^2, x, 1 \rangle$

The Kernel Trick

- Take original feature vectors (say in d dims.) and map them to $D \gg d$ dims.
- Use a non-linear map $\phi: \mathbb{R}^d \to \mathbb{R}^D$ so that we can hope that vecs. in D dims. are mostly linearly independent
- Use linear models (SVM, lin. reg., k-means, PCA, etc.) on these new, D dimensional feature vectors
- $\because$ ϕ is a non-linear map, our final classifier/regressor $w^T\phi(x)$, $w \in \mathbb{R}^d$ will look non-linear in the original feature vectors x even though
- Only catch is the running time
  Most ML algos take time $\Omega(dn)$ so the above scheme may take time $O(dDn)$ time to prepare the new vectors and then $O(Dn)$ time to execute the ML algos.
- $\because$ Earlier argument works for $D \geq n$, the above is $O(dn^2)$
- $\therefore$ Kernel trick allows us to, for some very special non-linear maps $\mathbb{R}^d \to \mathbb{R}^D$, run ML algos on the D-dim vecs. w/o ever calculating ϕ explicitly.

Kernels vs. Distance Measures

$K(x, y)$	$d(x, y)$
• Give measures of similarity	• — of dissimilarity
• High value ⇒ Similar points	• High value ⇒ Dissimilar
• Ex: Gaussian, polynomial	• Ex: Euclidean, Mahalanobis
• Nice similarity funcs. satisfy the Mercer's theorem	• ——— metric or norm properties
• Supports kNN, LWP and can be used to implement SVMs, Least Squares Regression, etc.	• — (multi-label) classif., regression via kNN/LWP, Clustering via k-means, etc.

Mercer Kernels

- Suppose $x, y \in \mathbb{R}^d$ are any two unit vectors
  → Dot product is a natural notion of similarity
  Highest when vectors are same, ie. $x = y \Rightarrow x^T y = 1$
  Lowest when vectors are diametrically opposite, ie, $x = -y \Rightarrow x^T y = -1$
  * Mercer Kernels are notions of similarity that extend such nice behaviour
- Given a set of objects $\mathcal{X}$ (images, videos, strings, genome sequences), a similarity function $K: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **Mercer Kernel** if

$$\exists \text{ map } \phi: \mathcal{X} \to \mathcal{H} : \forall\, x, y \in \mathcal{X},\ K(x,y) = \langle \phi(x), \phi(y) \rangle$$

- ϕ often called feature map or feature embedding.
  $\mathcal{H}$ can be $\mathbb{R}^D$ for some large / moderate D.
  $\mathcal{H}$ can even be infinite dimensional.
- Thus, when asked to give similarity b/w 2 objects, all that a Mercer Kernel does is first map those objects to 2 (high-dim) vectors and return their dot / inner products.

Examples ($x, y \in \mathbb{R}^d$ are vectors)

Linear Kernel : $K_{lin}(x,y) = \langle x, y \rangle$
Quadratic " : $K_{quad}(x,y) = (\langle x, y \rangle + 1)^2$
Polynomial " : $K_{poly}(x,y) = (\langle x, y \rangle + c)^p,\ p \in \mathbb{N}$
Gaussian " : $K_{gauss}(x,y) = e^{-\gamma \cdot \|x-y\|_2^2}$
Laplacian " : $K_{lap}(x,y) = e^{-\gamma \|x-y\|_1}$

- $c < 0 \Rightarrow$ Non-mercer
- $c \geq 0 \Rightarrow$ Mercer
  - $c = 0 \Rightarrow$ Homogeneous

- There do $\exists\ \phi$ for each of them (finding it is tedious)
- p, γ need to be tuned! Large values can cause overfitting
- All the above are notions of similarity, can be verified using $\|x\|_2 = 1 = \|y\|_2$: $x = y$ (Largest); $x = -y$ (Smallest);

Finding ϕ

- $K_{lin}(x, y) = \langle x, y \rangle \rightsquigarrow \phi(x) = x$
  Called "linear" because any linear func. over $\phi(x)$ is a linear func. over the original x
- $K_{quad}(x, y) = (\langle x, y \rangle + 1)^2$
  $(\langle x, y \rangle + 1)^2 = 1 + 2\langle x, y \rangle + \sum_{i,j}^{d} x_i x_j y_i y_j = \langle \phi(x), \phi(y) \rangle$
  when $\phi(x) = \langle \phi_0(x), \phi_1(x), \phi_2(x) \rangle \in \mathbb{R}^{1+d+d^2}$
  where $\phi_0(x) = [1] \in \mathbb{R}^1$, $\phi_1(x) = \sqrt{2} \cdot x \in \mathbb{R}^d$,
  $\phi_2(x) = [x_1 x_1, x_1 x_2, x_1 x_3, \ldots, x_1 x_d, x_2 x_1, \ldots, x_d x_d] \in \mathbb{R}^{d^2}$
- And similar constructions for polynomial kernels
- called "quadratic" because a linear func. over $\phi(x)$ is a quadratic func. over x
- If we use a linear ML algo over $\phi(x) \in \mathbb{R}^{1+d+d^2}$, can learn any quadratic func. over original features $x \in \mathbb{R}^d$. Polynomial kernel of degree p similarly allows learning of degree p polynomial funcs. over original data.

* Homogeneous poly. kernels use features only of the form $\prod_{k=1}^{d} x_k^{p_k}$ $(p_k \geq 0)$ and $\sum_{k=1}^{d} p_k = p$.
But, if we have $c > 0$ then the kernels use all features of the form $\prod_{k=1}^{d} x_k^{p_k}$, $p_k \geq 0$, $\sum_{k=1}^{d} p_k \leq p$.
Non-homo. poly. kernels use more expressive features

* There may exist more than one ϕ for the same K
Ex: $\tilde{\phi}(x) = [\phi_0(x), \tilde{\phi}_{11}(x), \tilde{\phi}_{12}(x), \phi_2(x)]$, $\tilde{\phi}_{11}(x) = x = \tilde{\phi}_{12}(x)$ for K_{quad}

- K_{gauss} / K_{lap} correspond to infinite dimensional maps

$$K_{gauss}(x, y) = e^{-\gamma \|x - y\|_2^2} = e^{-\gamma \|x\|_2^2} e^{-\gamma \|y\|_2^2} e^{2\gamma \langle x, y \rangle}$$
$$= e^{-\gamma \|x\|_2^2} e^{-\gamma \|y\|_2^2} \sum_{i=0}^{\infty} (2\gamma)^i \cdot \langle x, y \rangle^i / i!$$

Gaussian Kernel is an inf. LC of poly kernels of all orders. Let ϕ_i^{poly} be a map for the poly kernel $\langle x, y \rangle^i$. Then a map for K_{gauss} is

$$\phi_{gauss}(x) = e^{-\gamma \cdot \|x\|_2^2} \cdot \left[\frac{\sqrt{(2\gamma)^i}}{\sqrt{i!}} \cdot \phi_i^{poly}(x) \right]_{i=0}^{i=\infty}$$

* Learning a linear func. over the features $\phi_{gauss}(x)$ amounts to learning an inf. degree poly. over x. Gaussian / Laplacian kernels are very powerful kernels, often called **universal kernels**.
By using these kernels, theoretically speaking, one can learn any function over data.

Some Domain Specific Kernels

- When x, y are bag of words features for strings. Let dictionary $\Sigma = \{w_1, w_2 \dots w_d\}$. Let $c_i(x)$ be the count of word i in string x
Kernel based on intersection
$K_{int}(x,y) = \sum_{i=1}^{d} \min\{c_i(x), c_i(y)\}$ → Mercer Kernel
$K_{int\text{-}n}(x,y) = \sum_{i=1}^{d} \min\{c_i(x), c_i(y)\} / \sqrt{c_i(x)\, c_i(y)}$ {0% := 0} → Normalized Intersection
- More generally, when $X, Y \subseteq \mathcal{U}$ are sets
Intersection Kernel $K_{int}(X,Y) = |X \cap Y|$
Norm. Int. Kernel $K_{int\text{-}n}(X,Y) = |X \cap Y| / \sqrt{|X| \cdot |Y|} \in (0,1)$
The above are just the linear kernel in disguise and hence clearly Mercer. Represent the set X using an indicator vector $\mathbb{I}_X \in \{0,1\}^{|\mathcal{U}|}$ with $\mathbb{I}_X[i] = 1$ if $i \in X$ else $\mathbb{I}_X[i] = 0$. So, $|X \cap Y| = \langle \mathbb{I}_X, \mathbb{I}_Y \rangle$
- N-gram, substring. Fisher kernels: other kernels b/w two strings
- Random walk kernels b/w two graphs
- Subtree, convolutional kernels b/w two trees
- Pyramid kernel used in vision — combo. of ∩ kernels
- In practice, we often use a linear method first. eg. SVM / ridge regression. If that gives unsatisfactory

performance, we often jump directly to Gaussian kernel (although other kernels shouldn't be neglected). ∃ "kernel learning" methods that can learn the most appropriate kernel for us or else tune the kernel parameters eg. p, c, γ automatically.

Creating New Kernels

M1: Combine old kernels (K_1, K_2 say):

$K_3 := c_1 K_1 + c_2 K_2$, $c_1, c_2 \geq 0$;

$K_4 := K_1 \cdot K_2$; $K_5(x,y) = K_1(x,y) / \sqrt{K_1(x,x) \cdot K_1(y,y)}$ → Normalized Kernels

M2: Find a new feature rep. for data $\phi_{new}(x) \in \mathbb{R}^d$ and use $K_{new}(x,y) = \langle \phi_{new}(x), \phi_{new}(y) \rangle$

M3: Mix and match. Use $\phi_{new}(x)$ and K_{old}

$K_{new}(x,y) = K_{old}(\phi_{new}(x), \phi_{new}(y)) \equiv \langle \phi_{old}(\phi_{new}(x)), \phi_{old}(\phi_{new}(y)) \rangle$

Remarks:
- If K_1 gives very large values (in magnitude), some algo. may suffer.
- Normalized version of K_1, i.e., K_5, always gives values in $[-1, 1]$. It also normalizes the feature map as well. If ϕ is a map for K_1 & for K_5 is $\tilde{\phi}$, then $\tilde{\phi} : x \mapsto \phi(x) / \|\phi(x)\|_{\mathcal{H}}$

Kernelized Algorithms

- Several algos. that we've studied, work w/ kernels too
  - ↳ Supervised: kNN, LWP, SVM, ridge regression
  - ↳ Unsupervised: k-means, PCA
- Probabilistic / Bayesian algos. possible w/ kernels. Others like LASSO are harder to get working w/ kernels.

Special Case of Naïve Bayes Learning

→ Use a standard Gaussian $N(\mu_c, I_d)$ to model points of class c. MLE estimate of μ_c: mean of points of class c — a "prototype" of class c

- If we assume $p_1 = p_2 \ldots = p_C$, then to classify a test point x^t, simply find $\underset{c \in [C]}{\operatorname{argmin}} \|x^t - \hat{\mu}_c\|_2$
- Called the "Learning with Prototype" (LwP) model ← Linear Decision Boundary

→ Suppose we let every train point be its own cluster → kNN algorithm

- Let training set be $\{(x^i, y^i)\}_{i=1}^n$ where $y^t \in [C]$. Given test point x^t, find $i^t := \underset{i \in [n]}{\operatorname{argmin}} \|x^t - x^i\|_2$. Give y^{i^t} as the output — may consult more neighbours

★ · Many ML algos work even if feature vectors are not provided directly but instead pairwise dot/inner products b/w feature vectors is provided.

· For training, pairwise dot products b/w train points needed. For testing, <test point, train point> needed ∀ train points

· Thus, we can say we want to work w/ (high-dim) feature vectors $\phi(\cdot)$ and when ML algo asks for dot products, give it $K(x,y) = \langle \phi(x), \phi(y) \rangle$

↳ Get same results as when working directly w/ $\phi(x)$ but without having to compute ϕ

This peculiar property is often called **kernelizability**. An ML algo. is said to be kernelizable if it can be shown to work identically w/ feature vectors / pairwise train-train & test-train dot products of feature vectors

kNN with Kernels: All that is needed to execute kNN

is compute Euclidean distances. If working with kernel K with map ϕ, need $\|\phi(x)-\phi(y)\|_{\mathcal{H}}^2$. $\|\cdot\|_{\mathcal{H}}$ is just fancy notation for $\|\cdot\|_2$ in a Hilbert space $\mathcal{H}$.

$$\begin{aligned}\|\phi(x)-\phi(y)\|_{\mathcal{H}}^2 &= \|\phi(x)\|_2^2 - 2\langle\phi(x),\phi(y)\rangle + \|\phi(y)\|_2^2 \\ &= \langle\phi(x),\phi(x)\rangle - 2\langle\phi(x),\phi(y)\rangle + \langle\phi(y),\phi(y)\rangle \\ &= K(x,x) - 2K(x,y) + K(y,y)\end{aligned}$$

Thus, distances in $\mathcal{H}$ can be computed without computing ϕ first. K usually takes $O(d)$ time if $x, y \in \mathbb{R}^d$ but computing ϕ might take much longer (eg. Gaussian kernel: ∞ time)

1NN: Given training $(x^i, y^i)_{i=1}^n$ and a test data point x^t, find closest neighbour in $\mathcal{H}$, ie, $i^t = \operatorname{argmin}_{i\in[n]} \|\phi(x^t) - \phi(x^i)\|_{\mathcal{H}}^2$ which is the same as $\operatorname{argmin}_{i\in[n]} K(x^i,x^i) - 2K(x^i,x^t)$ and predict y^{i^t} as the label.

- Note: If $K(x,x) \equiv 1$ then this finds most similar point, ie, $\operatorname{argmax}_{i\in[n]} K(x^i,x^t)$. Similarly, we can execute kNN for $k>1$ as well.
- This is a recurring theme in kernel learning. NEVER ever compute $\phi(\cdot)$. Instead, express all operations in the algo in terms of inner products.
- So, kNN boils down to choosing a kernel K (w/ map ϕ) and computing $K(\cdot,\cdot)$.

LwP: Given train data (x^i, y^i), $y^i \in \{-1,+1\}$, we earlier found prototypes p^+ and p^- and used them to predict on test data point x^t as $\operatorname{sign}(\|x^t-p^-\|_2^2 - \|x^t-p^+\|_2^2)$

- If using a kernel K (w/ ϕ), we should now compute

new prototypes as $\tilde{p}^+ = \frac{1}{n^+} \sum_{y^i=1} \phi(x^i)$ and $\tilde{p}^- = \frac{1}{n^-} \sum_{y^i=-1} \phi(x^i)$

and predict using $\text{sign}\left(\|\phi(x^t) - \tilde{p}^-\|_2^2 - \|\phi(x^t) - \tilde{p}^+\|_2^2\right)$

- Can't compute these new prototypes explicitly; Reduce the above to kernel computations:

$$\|\phi(x^t) - \tilde{p}^+\|_2^2 = \underbrace{\langle \phi(x^t), \phi(x^t) \rangle}_{K(x^t, x^t)} - \underbrace{2 \langle \phi(x^t), \tilde{p}^+ \rangle}_{\frac{2}{n^+} \sum_{y^i=1} K(x^t, x^i)} + \underbrace{\langle \tilde{p}^+, \tilde{p}^+ \rangle}_{\frac{1}{n^+ n^+} \sum_{y^i=1} \sum_{y^j=1} K(x^i, x^j)}$$

can be pre-calculated at train time

- However, we now need to store entire training data (instead of two prototypes). This is common in kernel learning — larger model sizes and longer prediction times.

Kernel SVM

- Primal Formulation $\begin{cases} \min_{w, \xi_i} \ \frac{1}{2}\|w\|_2^2 + C \cdot \sum_{i=1}^n \xi_i \\ \text{s.t.} \ \ y^i \cdot w^T x^i \geq 1 - \xi_i \ \ \& \ \ \xi_i \geq 0 \end{cases}$

- Dual Formulation $\max_{\alpha_i} \ \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y^i y^j \langle x^i, x^j \rangle$ s.t. $\alpha_i \in [0, C]$

↓ After applying a non-linear feature map ϕ → If $\phi : \mathbb{R}^d \to \mathcal{H}$ then $w \in \mathcal{H}$

- Primal → $y^i \cdot w^T \phi(x^i) \geq 1 - \xi_i$ & $\xi_i \geq 0$

Dual → $\max_{\alpha_i} \ \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y^i y^j \underbrace{\langle \phi(x^i), \phi(x^j) \rangle}_{\equiv K(x^i, x^j)}$ s.t. $\alpha_i \in [0, C]$

- Primal is infeasible to solve.
  If $\dim(\mathcal{H}) = \infty$, a single SGD step takes ∞ time
  However, dual is feasible $\because K(x^i, x^j)$ usually takes $O(d)$ time
  Each step of SDCA takes $O(n)$ time apart from $K(x^i, x^j)$.
  Combining both yields $O(nd)$ time per step.
- Finding the model explicitly is not feasible even if we solve the dual problem perfectly
  $\because \ w = \sum_{i=1}^n \alpha_i y^i \phi(x^i) \in \mathcal{H}$ and $\dim(\mathcal{H}) \gg 1$
- Instead, we can store all the α_i values (n of them). Prediction: $w^T \phi(x^t) = \sum_{i=1}^n \alpha_i y^i K(x^i, x^t)$

- If x^t similar to some training points, i.e., $K(x^i, x^t)$ is large, then y^i influences the pred. much more.
- If there are $\tilde{n}$ support vectors, then prediction requires $\tilde{n}$ kernel computations, i.e., roughly $O(\tilde{n}d)$ time.

Kernel SVM looks like a "soft" form of kNN.

Kernel Ridge Regression

- Given data point $(x^i, y^i)_{i=1}^n$, $x^i \in \mathbb{R}^d$, $y^i \in \mathbb{R}$, RR solution is simply $\hat{w} = (X^T X + \lambda \cdot I_d)^{-1} X^T y$
- Dual for RR: $\min_{w \in \mathbb{R}^d} \lambda \cdot \|w\|_2^2 + \|y - Xw\|_2^2$
- Dual requires constraints — let's deliberately introduce some $\min_{w \in \mathbb{R}^d, z \in \mathbb{R}^d} \lambda \cdot \|w\|_2^2 + \|z\|_2^2 \text{ s.t. } z = y - Xw$
  Lagrangian becomes $\lambda \cdot \|w\|_2^2 + \|z\|_2^2 + \alpha^T (y - Xw - z)$
  ↳ FOO gives us $z = \alpha/2$ & $2\lambda \cdot w = X^T \alpha$
  Dual becomes → $\min_{\alpha \in \mathbb{R}^n} \frac{1}{4\lambda} \cdot \sum_{i,j} \alpha_i \alpha_j \langle x^i, x^j \rangle + \frac{\|\alpha\|_2^2}{4} - \alpha^T y$

Kernelizable → Solve $\hat{\alpha} = \operatorname{argmin}_{\alpha \in \mathbb{R}^n} \sum_{i,j} \frac{\alpha_i \alpha_j K(x^i, x^j)}{4\lambda} + \frac{\|\alpha\|_2^2}{4} - \alpha^T y$

Model: $\hat{w} = \sum_{i=1}^n \frac{\hat{\alpha}_i \cdot \phi(x^i)}{2\lambda} \in \mathcal{H}$ ↳ Can't be stored explicitly

Prediction: $\langle \hat{w}, \phi(x^t) \rangle = \frac{1}{2\lambda} \cdot \sum_{i=1}^n \hat{\alpha}_i \cdot K(x^i, x^t)$

- A simplification: Let $G = [K(x^i, x^j)]_{i,j} \in \mathbb{R}^{n \times n}$ be the "Gram matrix" of the training points.

Dual of RR: $\min_{\alpha \in \mathbb{R}^n} \frac{1}{4\lambda} \alpha^T (G + \lambda I_n) \alpha - \alpha^T y \Rightarrow \hat{\alpha} = 2\lambda (G + \lambda I_n)^{-1} y$

Requires inverting $n \times n$ matrix (linear RR required inverting $d \times d$ matrix) ← Closed form

Kernel Clustering: K-Means / Lloyd's Algorithm -

1. Init centroids $\{\mu^k\}_{k=1 \dots K} \in \mathbb{R}^d$
2. For $i \in [n]$, assign clusters, update $z^i \in [K]$ using μ^k: $z^i \leftarrow \operatorname{argmin}_k \|x^i - \mu^k\|_2^2$
3. Update $\mu^k \leftarrow \sum_{i: z^i = k} \frac{x^i}{n_k}$, $n_k := |\{i : z^i = k\}|$ 4. Repeat until convergence

All we need to do is kernelize the distance computations and keep track of the cluster centers implicitly ∵ now $\mu^k \in \mathcal{H}$

Kernel K-means

- cluster centers in k-means are always the average of data points that were assigned to that cluster — z^i maintains this info but need to maintain it a bit differently for easy processing
- Let $\alpha_i^k = 1$ if $z^i = k$ and $\alpha_i^k = 0$ if $z^i \neq k$, i.e., $n_k = \sum_{i=1}^n \alpha_i^k = 1^T \alpha^k \rightsquigarrow \mu^k = (\sum_{i=1}^n \alpha_i^k)^{-1} \cdot \sum_{i=1}^n \alpha_i^k \cdot \phi(x^i) \in \mathcal{H}$
- Let $G = [K(x^i, x^j)]_{i,j} \in \mathbb{R}^{n \times n}$ denote the Gram matrix of training points. Then, the distance computations can be rewritten as: $\|\phi(x^j) - \mu^k\|_{\mathcal{H}}^2 = K(x^j, x^j) - 2\langle \mu^k, \phi(x^j) \rangle + \langle \mu^k, \mu^k \rangle$
- **Algorithm:** 1. Init $\{\alpha^k\}_{k=1}^K \in \{0,1\}^n$ randomly
  2. For $i \in [n]$ assign clusters:
  2.1. $n^k \leftarrow 1^T \alpha^k$ 2.2. $z^i = \text{argmin}\ n_k^{-2} (\alpha^k)^T G \alpha^k - 2 n_k^{-1} G_i^T \alpha^k$
  3. For $k \in [K]$ update α^k: For $i \in [n]$, set $\alpha_i^{z^i} = 1$ & $\alpha_i^l = 0\ \forall\ l \neq z^i$
  4. Repeat until convergence

$(\sum_{i=1}^n \alpha_i^k)^{-1} \sum_{i=1}^n \alpha_i^k \cdot K(x^i, x^j)$ $(\sum_{i=1}^n \alpha_i^k)^2 \sum_{i,j=1}^n \alpha_i^k \alpha_j^k K(x^i, x^j)$

$= (1^T \alpha^k)^{-1} G_j^T \alpha^k$ $= (1^T \alpha^k)^{-2} (\alpha^k)^T G \alpha^k$

$\hookrightarrow j^{th}$ Column of G

- **Parametric ML models:** for which size is independent of # of training points. Ex: Linear SVM, LwP, Logistic regression, Ridge regression.
- **Non-parametric ML models:** —— dependent ——
  Ex: Kernel SVM w/ non-linear kernel, kNN.
  - Kernel algorithms in general are non-parametric.

Linear Algebra

Vectors are a list of real numbers /integers that describe features / quantities.

Linear Combination of Vectors

- Given two vectors $a, b \in \mathbb{R}^d$ and scalars $p, q \in \mathbb{R}$, we can $p \cdot a \pm q \cdot b$ (scale them; add/subtract them)
- $p \cdot a$, p varies: line. $p \cdot a + q \cdot b$, p varies: line. $p \cdot a + q \cdot b$, p & q vary: plane (unless $d \cdot a = c \cdot b$ for some d, c)
- Convex combinations: $p, q \geq 0$; $p + q = 1$; Line segment
- Affine combinations: $p + q = 1$; entire line
- Linear combinations: No restrictions on p, q. Could be the whole $\mathbb{R}^2$ unless $\exists\ c : a = c \cdot b \rightsquigarrow (cp + q)b$
- More than two vectors: Given n points $x^1, x^2 \ldots x^n \in \mathbb{R}^d$, a convex combination is obtained by selecting $p_1, \ldots, p_n$ s.t. $p_i \geq 0$ & $\sum p_i = 1$ and computing $\sum p_i \cdot x^i$. The set of all convex combinations gives us the convex hull of these points.
- Span of more than two vectors: Span of: 1 vector x^1: a line, add x^2: plane, add x^3: space...

Dependence & Independence: If we can express x^n as an LC of some other vectors $x^1 \ldots x^{n-1}$, then we say that x^n is Linearly Dependent on $x^1 \ldots x^{n-1}$: x^n in such cases is redundant

$$\text{span}(x^1 \ldots x^n) = \text{span}(x^1 \ldots x^{n-1})$$

- A set of vectors $x^1 \ldots x^n$ is said to be Linearly Independent: if no vector x^i can be written as an LC of the other vectors $\{x^j\}_{j \neq i}$

Basis: Given a set of vectors $S = \{x^i\}$, consider the set span(S). A set of vectors B is called a

basis for the set S if vectors in the set B are LI as well as $\text{span}(B) = \text{span}(S)$. Always possible to choose $B \subseteq S$.

- LI assures that the basis cannot be shrunk further.
- If vectors in B are orthogonal, i.e., $b^1, b^2 \in B,\ b_1 \neq b_2 \Rightarrow b_1 \perp b_2$. then B is called an **orthogonal basis**.
- If vectors in B are orthonormal, ie, orthogonal as well as $\|b\|_2 = 1\ \forall\ b \in B$. then B is called an **orthonormal basis**.
- These definitions hold true even if S is infinite

Gram-Schmidt Orthonormalization Process (GSO)

Given $S = \{x^1 \dots x^n\}$, this process yields an orthonormal basis of S:

▷ Init basis by an element x^i from S: $B = \{x^i / \|x^i\|_2\}$
▷ while $(S \neq \phi)$: ▷ Remove an arbitrary element from S, say x ▷ Compute $\tilde{x} = x - \sum_{b \in B} (b^T x) \cdot b$ ▷ If $\tilde{x} \neq 0$, add $\tilde{x} / \|\tilde{x}\|_2$ to B else throw $\tilde{x}$ away

- We will get a basis of the same size every time.
- Can be numerically imprecise, but $\exists$ stable versions

Linear Maps / Transformations (LT)

- These map vectors to other vectors. but in a way that preserve lines. Always maps the origin to origin itself.
- If a map preserves all lines but shifts the origin, it is called an **affine transformation**
- Mathematically, function $f: \mathbb{R}^d \to \mathbb{R}^k$ is a linear map / transformation if:

$f(x+y) = f(x) + f(y) \quad \forall\ x, y \in \mathbb{R}^d$ & $f(c \cdot x) = c \cdot f(x) \quad \forall\ c \in \mathbb{R}, \forall\ x \in \mathbb{R}^d$

($x+y \in \mathbb{R}^d$; $f(x), f(y) \in \mathbb{R}^k$; take $c=0$ then $0 \in \mathbb{R}^d$ mapped to $0 \in \mathbb{R}^k$)

Encoding LTs: Let $e_i = (0, 0 \dots 0\,(1)\,0 \dots 0) \in \mathbb{R}^d$ (i^{th} entry) → e_i: i^{th} Canonical/Standard/Natural basic vector

- Suppose we know $f(e_1), f(e_2) \dots, f(e_d)$ $(\in \mathbb{R}^k)$
- For any new $v = (v_1 \dots v_d) \in \mathbb{R}^d$, we can use linearity to find $f(v) = f(\Sigma v_i e_i) = \Sigma f(v_i e_i) = \Sigma v_i f(e_i)$

Matrices = LTs: Arrange all these d vectors (all k-dim.) as column vectors side by side, we get a $k \times d$ matrix $A \in \mathbb{R}^{k \times d}$: $A = \begin{bmatrix} | & | & & | \\ f(e_1) & f(e_2) & \dots & f(e_d) \\ | & | & & | \end{bmatrix}$ then $\Sigma v_i f(e_i)$ is the same as Av.

- Thus, all LTs $\mathbb{R}^k \to \mathbb{R}^d$ can be expressed as a matrix $A_{k \times d}$ s.t. Av gives the LT. Easy to verify that for any matrix $B \in \mathbb{R}^{k \times d}$, the transformation defined as $g: v \mapsto Bv$ is always linear.

Special LTs

- **Scaling Transformations:** Scale the various dimensions by different amounts. Output vector has same dimensions
- Sometimes, scaling by zero can be treated as equivalent to dropping a dimension.
- These correspond to diagonal matrices.

- **Rotation Transformations:** Also called "isometric transformations"
- Simply rotate the whole space. Also preserve the dot product b/w any two vectors. Output vector has same dimensions.
- Always correspond to matrices whose columns are orthonormal. (Not the other way around though, ∵ orthonormal matrices can flip the sign of an axis or exchange them)

* If the vectors $f(e_1) \dots f(e_d)$ are not LI, then the entire space is mapped to a lower-dimensional hyperplane / plane.
  This could happen even if k (the output dim.) is a large number, even if $k \gg d$.

☆ The possible values of Av are exactly $\text{span}(f(e_1) \dots f(e_d))$

· **Applying Multiple LTs:** Let $f: \mathbb{R}^d \to \mathbb{R}^k$ and $g: \mathbb{R}^k \to \mathbb{R}^l$
  ↳ $A \in \mathbb{R}^{k \times d}$ ↳ $B \in \mathbb{R}^{l \times k}$
  The transformation $h: x \mapsto g(f(x))$ is then simply $B(Ax) = (BA)x$, ie, h is represented by the matrix $C = BA$.

The **Universal LTs**: It turns out that scaling and rotation transformations are all we need to understand in order to understand ANY LT.

* Some natural interpretations as linear maps:
  Let $f, g: \mathbb{R}^d \to \mathbb{R}^d$ be maps w/ matrices $A, B \in \mathbb{R}^{d \times d}$
  · Addition: $A+B \equiv p(x) := f(x) + g(x) \quad \forall x$
  · Multiplication: $AB \equiv q(x) := f(g(x)) \quad \forall x$
  · Identity Matrix: corresponds to the map $i(x) = x \ \forall x$
  · Scaling: $c \cdot A$ corresponds to the map $r(x) := c \cdot f(x) \ \forall x$
  · Inverse: A^{-1} corresponds to $s: \mathbb{R}^d \to \mathbb{R}^d$ s.t. $f(s(x)) = s(f(x)) = i(x) = x \ \forall x$
  · Let $h: \mathbb{R}^d \to \mathbb{R}^k$ w/ $C \in \mathbb{R}^{k \times d}$.
  Matrix Transpose: C^T is a map $t: \mathbb{R}^k \to \mathbb{R}^d$. No relation w/ C in general, just another map. For symmetric matrices however, transpose is the same as the original one.

* $(AB)^{-1} = B^{-1}A^{-1}$; $(A+B)^{-1} \neq A^{-1} + B^{-1}$; $c(A+B) = cA + cB$; $A+B = B+A$; $(c+d) \cdot A = cA + dA$

· A vector (column by default) $v \in \mathbb{R}^{n = n \times 1}$ is a map $\mathbb{R} \to \mathbb{R}^n$ which maps a scalar $c \in \mathbb{R}$ to a vector $c \cdot v \in \mathbb{R}^n$

· A row vector $u^T \in \mathbb{R}^{1\times n}$ is a map from $\mathbb{R}^n \to \mathbb{R}$
It maps a vector $v \in \mathbb{R}^n$ to a scalar $u^T v \in \mathbb{R}$
So, inner/dot products can be seen as a linear map.

Outer product: Given a col. vector $v \in \mathbb{R}^m$ & row vector $u^T \in \mathbb{R}^{1\times n}$, can think of vu^T as a composition map from $\mathbb{R}^n \to \mathbb{R} \to \mathbb{R}^m$, ie, from $\mathbb{R}^n \to \mathbb{R}^m$.
→ $(vu^T)x = (u^T x)v \in \mathbb{R}^m$ and $U = vu^T \in \mathbb{R}^{m\times n}$: matrix
— Example of non-comm. matmul: $UU^T = (vu^T)(uv^T) = (u^Tu)\cdot vv^T$ while $U^TU = uv^Tvu^T = (v^Tv)\,uu^T$ might not even have the same dimensions if $m \neq n$.

Column Space

· As $v \in \mathbb{R}^d$ takes all possible values, $Av = \sum v_i f(e_i)$ takes all possible values in $\text{span}(f(e_1) \dots f(e_d))$
— called the **column space** of A. It tells us all all possible outputs that a map can give.

Rank: Given a set of vectors S, the number of vectors in its basis is called the rank of the set S
— often written as rank(S) or even dim(S)

Special Case 1: $\dim(S) = 0$: Only if the set contains only one element (or copies of it) & that element is 0.
SC2: $\dim(S) = 1$: All vectors in S all lie on a line passing through the origin, even if dims. $d > 1$
SC3: $\dim(S) = 2$: All lie on a 2D plane passing origin.
SC4: $\dim(S) = k$: All vectors lie on a k-dim. subspace

· $\dim(S) = \dim(\text{span}(S)) \leq |S|$
· The lines that do not pass through origin must be dealt w/ carefully. Ex: Rank of a line not through origin is 2, not 1. Verify with $x+y=1 \to (0,1), (1,0) \to \text{span}(x+y=1) = \mathbb{R} \Rightarrow \text{rank} = 2$

Column Rank: For a matrix A, rank of its set of columns is called column rank of A. Rank of its set of rows is called row rank of A.

Claim: Row rank & column rank of a matrix are always the same. So, simply talk about the rank of A.

* A matrix $A \in \mathbb{R}^{m \times n}$ for which rank is equal to the # of rows/columns, i.e., $\text{rank}(A) = \min\{m, n\}$ is called a **full-rank matrix**. A matrix that is not full rank is called **rank-deficient**

Spaces: Simply a set of vectors that is self-contained in two ways: Take a / → Scale it → Resultant vector is two vector/s → Add them already there in space

· A space is closed w.r.t. addition & scalar multiplication.

* Zero vector must be inside every space S.

· Spaces can contain other spaces (called subspaces).

· **Affine Spaces**: A shifted (sub) space. Suppose $S \subseteq \mathbb{R}^d$ is a space. Then, for any $v \in \mathbb{R}^d$, the set $T_v := \{v + x : x \in S\}$ is an affine space. Every space is affine too (take $v \in S$)

· **Hyperplane**: in $\mathbb{R}^d$ is a subspace of rank $d-1$. All hyperplanes look like $\{x : w^T x = 0\}$ for some $w \in \mathbb{R}^d$

· **Affine Hyperplane**: A displaced hyperplane → $\{x : w^T x + b = 0\}$ for some $w \in \mathbb{R}^d$, $b \in \mathbb{R}$; $w^T x = -b$ iff $(x = v - wb/\|w\|_2^2)$ for some v s.t. $w^T v = 0$

Null Space: For any linear map w/ matrix A, its null space (aka kernel) is the set of inputs that get mapped to zero vector $\ker(A) = \{x \in \mathbb{R}^d : Ax = 0\}$ — subspace of $\mathbb{R}^d$

· The rank/dimension of $\ker(A)$ is called the nullity of A.

· $\dim(\ker(A)) = 0$ → A only maps 0^d to 0^k.

$= 1$ → A maps an entire line passing through origin to 0.

Similarly, for $\dim(\ker(A)) > 1$ $Au = A(u+v)$ $\forall\, v \in \ker(A)$ $\forall\, u$

Also, $Au = A\tilde{u} \Rightarrow A(u - \tilde{u}) = 0 \Rightarrow u - \tilde{u} \in \ker(A)$

Rank-Nullity Theorem: $\dim(\ker(A)) + \text{rank}(A) = d$

Orthonormal Matrices: Whose columns are unit L2 norm and $\perp$ to each other. If a column is multiplied by -1, it amounts to flipping that axis. If columns are shifted around, it amounts to exchanging/reordering them.

* $A^TA = I \Rightarrow A$ is invertible w/ A^T as its inverse. Also, A^T is also orthonormal and $A^TA = I$

Singular Value Decomposition (SVD)

- Let $A \in \mathbb{R}^{d \times d}$ be any (possibly non-symmetric) square matrix. Then we can always write A as a product of three matrices $A = U\Sigma V^T$ where $U, V \in \mathbb{R}^{d \times d}$ are orthonormal matrices & Σ is a scaling (diagonal) matrix w/ all entries non-negative. Thus, every matrix is simply ROT + SCA + ROT
- Diagonal elements in Σ are called singular values of A (Always ≥ 0)
  Column vectors of U are called the left singular vectors of A
  .. .. .. V .. .. .. right singular vectors of A
- If $U = [u^1 \dots u^d]$, $V = [v^1 \dots v^d]$, $\Sigma = \text{diag}(\sigma_1, \dots \sigma_d)$, then:
  $A = \sum_{j=1}^{d} \sigma_j \underbrace{u^j (v^j)^T}$ → Rank one matrix

Singular values of a matrix are always unique, singular vectors are not. In order to minimize ambiguity, people commonly write SVD s.t. $\Sigma = \text{diag}(\sigma_1 \dots \sigma_d)$ & $\sigma_1 \geq \dots \geq \sigma_d$

- SVD is defined even for matrices that are not square
  $A \in \mathbb{R}^{m \times n} \Rightarrow A = U\Sigma V^T$, $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$, $\Sigma^{m \times n}$, $\Sigma_{ii} \geq 0$, $\Sigma_{ij} = 0, i \neq j$

Case 1: $n > m$ i.e. output dim < input $\Rightarrow A$ reduces dim.
Σ throws out last $(n-m)$ dimensions after rotation by V^T

Case 2: $n < m$ i.e. $A \uparrow$ dim. $\longrightarrow \Sigma$ adds $(n-m)$ dummy dimensions in addition to scaling.

· **Rank**: We always have $\text{rank}(A) = \#(\Sigma_{ii} > 0)$ ⇝ If some diagonal entries of Σ are zero, we can remove those rows and corresponding columns of U.

If $A \in \mathbb{R}^{m \times n}$ and only r entries of Σ are non-zero, then we can equivalently write A as $A = \tilde{U}\tilde{\Sigma}V^T$ where $\tilde{U} \in \mathbb{R}^{m \times r}$, $\tilde{\Sigma} \in \mathbb{R}^{r \times n}$. This is often called the **thin SVD** of A. Finally $Av = \tilde{U}z$ where $z = \tilde{\Sigma}V^T v \in \mathbb{R}^r$ and thus, the column space of A is the column span of $\tilde{U}$ and clearly $\dim(\tilde{U}) = r$.

This also shows that row rank & column rank are same.

Trace & Determinant: Defined only for square matrices

Trace $= \text{tr}(A) := \sum A_{ii}$. Now, $\text{tr}(PQ) = \text{tr}(QP)\ \forall\ P, Q \in \mathbb{R}^{d \times d}$ as well as $\text{tr}(c \cdot P + Q) = \text{tr}(c \cdot P) + \text{tr}(Q) = c \cdot \text{tr}(P) + \text{tr}(Q)$

$\therefore \text{tr}(A) = \text{tr}\left(\sum \sigma_i u^i (v^i)^T\right) = \sum \sigma_i \,\text{tr}(u^i (v^i)^T) = \sum \sigma_i \langle u^i, v^i \rangle$

Determinant $= |\det(A)| = \prod \sigma_j$. Sign of $|A|$ depends on # of axes flipped/switched by U, V.

Inverses: A square matrix is invertible iff all its singular values are non-zero

· A singular value being zero means A squishes vectors along some direction to the origin. So, A's action cannot be undone ⇒ A is invertible iff $\det(A) \neq 0$

· If $A = U\Sigma V^T$ invertible, then we have $A^{-1} = V\Sigma^{-1}U^T$

Pseudo Inverse: Even if $A \in \mathbb{R}^{m \times n}$ is not invertible / square, we can still define the Moore-Penrose inverse of A:

$A^+ = V\Sigma^+ U^T \in \mathbb{R}^{n \times m}$. If $\Sigma = \text{diag}(\sigma_1 \dots \sigma_k) \in \mathbb{R}^{m \times n}$, $k = \min\{m, n\}$, we define $\Sigma^+ = \text{diag}(\tilde{\sigma}_1 \dots \tilde{\sigma}_k) \in \mathbb{R}^{n \times m}$ where $\tilde{\sigma}_i = 1/\sigma_i$ if $\sigma_i > 0$ else $\tilde{\sigma}_i = 0$

★ A^+ satisfies some nice properties:

- A invertible $\Leftrightarrow A^+ = A^{-1}$.
- AA^+ acts as an identity for its columns $\Rightarrow AA^+Az = Az \;\forall z$
- If $u = Az$, then $v = A^+u$ is s.t. $Av = u = Az$.
  Can be shown that v is the vector w/ smallest L2 norm s.t. $Av = u$, i.e., A^+ gives us the least squares solution. If $u = Az$, then $A^+u = \underset{x \text{ s.t. } Ax=u}{\operatorname{argmin}} \|x\|_2^2$
- If X^TX invertible, then $X^+ = (X^TX)^{-1}X^T$.
  Even if $\nexists z$ s.t. $Az = u$, even then, A^+u returns the vol^n w/ smallest error, i.e., $A^+u = \underset{x}{\operatorname{argmin}} \|Ax - u\|_2^2$, and among all vectors w/ this smallest error, A^+u is the one w/ smallest L2 norm.

Eigenvectors & Eigenvalues

- Let $X \in \mathbb{R}^{n \times d}$ be a feature matrix — want to find its SVD. Instead, consider $X^TX = V\Sigma^TU^TU\Sigma V^T = V\Sigma^T\Sigma V^T = V\Lambda V^T$
  This matrix is square, symmetric. Focus on finding V.
  Let $V = [v^1 \dots v^d]$, then $X^TX v^j = V\Lambda V^T v^j = V\Lambda e^j = V\lambda_j e_j = \lambda_j v^j$,
  $\Lambda = [\lambda_1, \dots \lambda_n]$. So, X^TX only scales the right sing. vectors of V.
  For U, consider $XX^T = U\tilde{\Lambda}U^T$

Eigenpair: (v, λ) for matrix A s.t. $Av = \lambda v$. λ could be negative or zero. Inf. no. of eigenpairs in general, $\because (v, \lambda) \Rightarrow (cv, c\lambda) \;\forall c \in \mathbb{R} \rightsquigarrow$ Define eigenvectors to be only unit vectors.

* Square roots of eigenvalues of XX^T give us all the non-zero singular values of X.

Definiteness: Note that $\forall x \in \mathbb{R}^d$ we have $x^TX^TXx = \|Xx\|_2^2 \geq 0$
ie, X^TX is a PSD (Positive Semi-Definite matrix)

* Every square symmetric matrix A can be written as $A = Q^TSQ$ where $Q \in \mathbb{R}^{d \times d}$ is orthonormal & S is diagonal

Write A as $A = \sum s_j q^j (q^j)^T \rightsquigarrow (q^j, s_j)$ are eigenpairs of A

Thus, $\forall\ x \in \mathbb{R}^d$, we have $x^T A x = \sum s_j (x^T q^j)^2$

If all $s_j \geq 0$ then A is a PSD. If even one $s_j < 0$, take $x = q^j$ → A is not PSD

- XX^T / X^TX is always a PSD.
- A sq. matrix has one or more zero eigenvalues iff it is not a full rank matrix

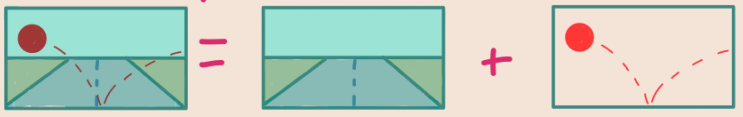
SVD v/s ED

- eigenvalues +/-/0 while singular values ≥ 0
- $\exists$ SVD $\forall$ (even rect.). $\exists$ ED $\forall$ sq. symm., might exist for non-symm.
- ED's do not make sense for rectangular matrices.
- For a PSD, ED = SVD.
- For a non-PSD sq. symm. matrix, ED can be used for SVD
  - ED: $A = QSQ^T$, $N = \text{diag}(\text{sign}(s_1) \dots \text{sign}(s_d))$, $U := QN$, $V := Q$, $\Sigma = |S|$
  - → $A = U\Sigma V^T$
- Sq. symm. matrices → Real eigenvalues. Others, no guarantee
- Rotation matrices in general do not have any eigenvectors ∵ they rotate every vector so no vector is transformed w/ just a simple scaling.
  - No ED → They do have an SVD → $A = AII$. For identity matrix, every vector is its eigenvector → SVD = ED; $I = III$

Principal Component Analysis (PCA)

- **Noise Removal**: If data features are d-dim. but data is really lying in a $k \ll d$ dim subspace, then it might be noise that is making the data d-dim.
- PCA can help extract the important k features: Given $X \in \mathbb{R}^{n \times d}$ compute PCA $\hat{X} = \hat{U}\hat{\Sigma}\hat{V}^T$, and use $\tilde{X} = \hat{U}\hat{\Sigma} \in \mathbb{R}^{n \times k}$ as a set of new k-dim features for the n data points.

  Training algos get sped up if k-dim used, testing may not ∵ we still need to compute $\hat{V}^T x^t$
- **Foreground-Background Separation**



- Make every frame a vector $x^i \in \mathbb{R}^d$, d: no. of pixels
- We have n frames: $X = [x^1 x^2 \dots x^n]^T \in \mathbb{R}^{n \times d}$; Background is a constant vector $b \in \mathbb{R}^d$. Let $1 = [1\ 1 \dots 1]^T \in \mathbb{R}^n$; Foreground treated as noise $f^i \in \mathbb{R}^d$. Let $F = [f^1 f^2 \dots f^n]^T \in \mathbb{R}^{n \times d}$
- This gives $X = 1 \cdot b^T + F$, ie, if noise is not too much, then X is approximately rank 1. Can do PCA & recover F as noise.

- Learning Prototypes: Given $X \in \mathbb{R}^{n \times d}$, compute PCA $\hat{X} = \hat{U}\hat{\Sigma}\hat{V}^T$. Notice that $\hat{V} \in \mathbb{R}^{k \times d}$ i.e. we can think of V as a dataset of k prototypes. All points in dataset X can be approximated well as an LC of these k prototypes — the LC given by $\hat{U}\hat{\Sigma}$.
- Specifically, i^{th} data point (i^{th} row of X) can be approximated as $x^i \approx \sum_{j=1}^{k} \hat{U}_{ij} \sigma_j \cdot v^j$

- Eigenfaces — given face images, the prototypes given by the leading few right singular vectors

- Latent Semantic Analysis (LSA/LSI)
  - Given n documents, each as bag-of-words representation w/ dictionary size d as very large $\approx 10^6$, discover $k \ll d$ "topics" about which the docs. are talking

  - Each topic represented by a prototypical doc. for that topic.
  - If $|U_{ij}| >> 1$ then j^{th} topic is really core to i^{th} doc.

- Recommendation Systems: Popular technique is "Collaborative Filtering".
- Have data for n users and their interactions w/ d items. For these n users, predict other items that they would also like. Done by discovering k "user types" & representing each user as a combination of these user types.
- Drawback of collab. filtering is that it gets more difficult to add new users to the system.
- A recsys where users have features, is called "content-based filtering".

- The largest singular value of a matrix is called its leading singular value. Similarly, eigenvalue.
- Corresponding left singular vector is called leading left singular value. Similarly, right sing. vector / eigenvector.
  There may be more than one singular vector w/ the same singular value.
- **PCA**: Process of finding the top few singular values and corresponding sing. vectors.
  Given $X \in \mathbb{R}^{n \times d}$ (assume $d < n$) w/ SVD $X = \tilde{U}\tilde{\Sigma}\tilde{V}^T$ where $\tilde{U} \in \mathbb{R}^{n \times d}$ $\tilde{\Sigma} = \text{diag}(\sigma_1, \dots \sigma_d)$ w/ $\sigma_1 \geq \dots \sigma_d \geq 0$ and $\tilde{V} \in \mathbb{R}^{d \times d}$. $\tilde{U}$ & $\tilde{V}$ both have orthonormal columns, $\tilde{V}$ sq. but $\tilde{U}$ not $\because$ we dropped the last $(n-d)$ columns of U which were useless, to get $\tilde{U}$
- Want to find the **leading triplets**, ie, $\{\sigma_i, u^i, v^i\}$ for $i = 1 \dots k$ **for some $k \leq d$**. The rows of $\tilde{U}$ are not necessarily orthonormal, ie, maybe $\tilde{U}\tilde{U}^T \neq I$
- Leading right singular vector of $X \equiv$ leading eigenvector of X^TX
  Left of $X \equiv$ — of XX^T
- Denote $A = X^TX = V\Lambda V^T = \sum \lambda_i v^i (v^i)^T$; $\lambda_i = \sigma_i^2$ & that we reorder things so that $\lambda_1 \geq \lambda_2 \geq \dots$.
  Assume for sake of simplicity that $\lambda_1 \geq \lambda_2 \Delta$ for $\Delta > 1$
- Sometimes called an (leading) eigengap — can be expressed additively too — $\lambda_1 \geq \lambda_2 + \epsilon$ for $\epsilon > 0$.
  Shall handle $\Delta = 1$ later.
- Note that $A^2 := AA = V\Lambda^2 V^T$ where $\Lambda^2 = \text{diag}(\lambda_1^2, \lambda_2^2 \dots)$
- Similarly, $A^s = V\Lambda^s V^T$ for any $s \geq 0$
- $\lambda_1 \geq \lambda_2 \Delta \Rightarrow \lambda_1^s \geq \lambda_2^s \cdot \Delta^s \Rightarrow \lambda_1^s \geq \lambda_j^s \cdot \Delta^s \quad \forall\ j \geq 2 \quad \forall\ s \geq 1$
  With **large s**, the **leading eigenvalue** **stands out.**
- Let $x \in \mathbb{R}^d$, $\alpha = V^Tx \rightsquigarrow x = \sum \alpha_j v^j \rightsquigarrow \because V$ orthonormal, we have $VV^Tx = x \rightsquigarrow A^sx = V\Lambda^sV^Tx = V\Lambda^s\alpha = \alpha_1 \lambda_1^s v^1 + \dots$

Power Method

$\Rightarrow A^s x \approx \alpha_1 \lambda_1^s v^1 \rightsquigarrow \hat{v}^1 := A^s x / \|A^s x\|_2 \approx v^1$

↳ Find $\hat{v}^1$, then use (λ^1, v^1) eigenpair of $X^T X$ to get $\hat{\lambda}_1 := \|X^T X \hat{v}^1\|_2 \approx \lambda^1$

→ x s.t. $\alpha_1 = x^T v^1 \neq 0$. If $\alpha_1 = 0$ then $\alpha_1 \lambda_1^s = 0$

↳ The larger the s, the better the approximation.

↳ can be initialized randomly, eg., $N(0, I)$

→ $A^s x$ in $O(sd^2)$ time using s mat-vec mults instead of A^s in first calculating $O(sd^3)$

→ To obtain $\|\hat{v}^1 - v^1\|_2 \leq \epsilon$, takes $s = O(d \log 1/\epsilon)$.

→ To find smaller eigenpairs, we "peel off" the largest eigenpair & repeat the power method

Peeling Method: 1. Input: Sq. Sym. Matrix $A \in \mathbb{R}^{d \times d}$ 2. Init $A^0 \leftarrow A$ 3. For $j = 1 \ldots k$ 3.1. $(\hat{\lambda}_j, \hat{v}_j) \leftarrow$ Pow-Method(A^{j-1}) $O(kd^2s)$ 3.2. $A^j \leftarrow A^{j-1} - \hat{\lambda}_j \cdot \hat{v}_j (\hat{v}_j)^T$ 4. Return $\{\hat{\lambda}_j, \hat{v}_j\}_{j=1}^k$

Power Method: 1. Input: Sq. Sym. Matrix $A \in \mathbb{R}^{d \times d}$ $O(sd^2)$ 2. Init: $x^0 \sim N(0, I)$ [or other random value] 3. For $t = 1 \ldots s$ 3.1. $z^t \leftarrow A x^{t-1}$ 3.2. $x^t = z^t / \|z^t\|_2$ 4. $\hat{v}^1 = x^s$ 5. $\hat{\lambda}_1 = \|A\hat{v}^1\|_2$

Normalize to prevent overflow

- Residue might still be left due to inaccurate estimation of λ_j, v^j

$A = \lambda_1 v^1 (v^1)^T + \lambda_2 v^2 (v^2)^T + \lambda_3 v^3 (v^3)^T + \ldots$

- $\underbrace{X}_{n \cdot d \text{ space}} = U \Sigma V^T \rightarrow$ find top $k \leq d$ $\rightarrow \underbrace{\hat{X}}_{\text{can be stored in } (n+d+1) \cdot k \text{ space}} = \hat{U} \hat{\Sigma} \hat{V}^T$ ($n \times k$, $k \times k$, $k \times d$)

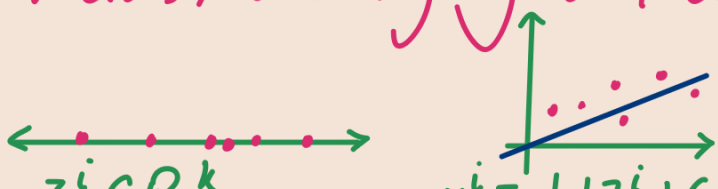
singular triplets

- $\hat{X} = \underset{\substack{Z \in \mathbb{R}^{n \times d} \\ \text{rank}(Z) \leq k}}{\text{argmin}} \|X - Z\|_F^2 \rightarrow \hat{x}$ best approximation of X among all rank-k matrices

$\|X\|_F$: Frobenius norm, stretch X into a long vector & take its L2 norm or take L2 norm of the vector formed by the singular values $\|X\|_F^2 = \sum_{ij} X_{ij}^2 = \sum_i \sigma_i^2$

Low Dimension Modelling: we may suspect that

our data features, although presented as d-dim. vectors, are lying on / close to some k-dim. subspace.



Given $x^1 \dots x^n$, can we recover $W, z^1 \dots z^n$?

$W_{d\times k}$ & $Z_{n\times k}$ s.t. $X \approx ZW^T$ → Given $X_{n\times d}$, recover

Dictionary / Factor Loading Matrix

→ Find $\hat{U}, \hat{V}, \hat{\Sigma}$ and set $Z = \hat{U}\hat{\Sigma}$ & $W = \hat{V}$, it will give us the best possible approx. any W, Z w/ only k columns could have given.

- **Regression v/s. PCA**: Most important difference is that in linear regression, features are visible, but in low-rank modelling setting, they are absent
- Latent variable modelling (AltOpt, EM) can indeed be used too. (latent)

Shortcomings of PCA: Will reveal hidden structure w/ in data if that hidden structure is a linear subspace. Fails to reveal it if data is lying on a curved (hyper) surfaces.

- May also fail if data is lying on an affine subspace ⇝ can be overcome by subtracting mean → $\mu = X^T 1/n$ & do PCA w/ $\tilde{X} = X - 1\mu^T$

- **Space Savings**

$X_{n\times d}$ features — $O(nd)$ space. Perform PCA & approx X using top k singular pairs. Takes $O((n+d)\cdot k) \ll nd$ space. Store $\hat{U}, \hat{V}, \hat{\Sigma}$ instead of $\hat{X}$. Applying $\hat{X}w = \hat{U}(\hat{\Sigma}(\hat{V}^T w))$ also takes $O((n+d)\cdot k) \ll nd$ time. How to choose k?

Time / Space budget: Choose k small enough s.t. $O((n+d)\cdot k)$ manageable

Error Tolerance: Choose k large enough s.t. $\|X - \hat{X}\|_F$ is acceptable

- $X - \hat{X} = \sum_{j=k+1}^{d} \sigma_j u^j (v^j)^T$; $\|A\|_F^2 = tr(AA^T) = tr(A^TA)$

$\Rightarrow \|X - \hat{X}\|_F^2 = tr\left(\sum_{j=k+1}^{d} \sigma_j^2 v^j (u^j)^T u^j (v^j)^T + \sum_{j \neq l} \sigma_j \sigma_l v^j (u^j)^T u^l (v^l)^T\right)$

- Now u^j are orthonormal, so are v^j, ie, $(u^j)^T u^l = 0$ & $(u^j)^T u^j = 1$ $\therefore \|X - \hat{X}\|_F^2 = tr\left(\sum_{j=k+1}^{d} \sigma_j^2 v^j (v^j)^T\right)$

$tr(v^j(v^j)^T) = tr((v^j)^T v^j) = 1$ $= \sum_{j=k+1}^{d} \sigma_j^2 \, tr(v^j (v^j)^T) = \sum_{j=k+1}^{d} \sigma_j^2$

- Similarly, can show that $\|X\|_F^2 = \sum \sigma_j^2$

A prominent knee point, if it exists, can give us a good idea of the true intrinsic dimensionality of data

☆ PCA minimizes reconstruction error and preserves maximum data variance.

Deep Learning (DL)

- From Kernel Learning to Deep Learning

No. of ← features very large → slow speed

Given a budget k, can we directly learn k good features? Deep Learning does exactly this. Instead of learning good features first and then learn a linear model on top of those features, DL does both jointly.

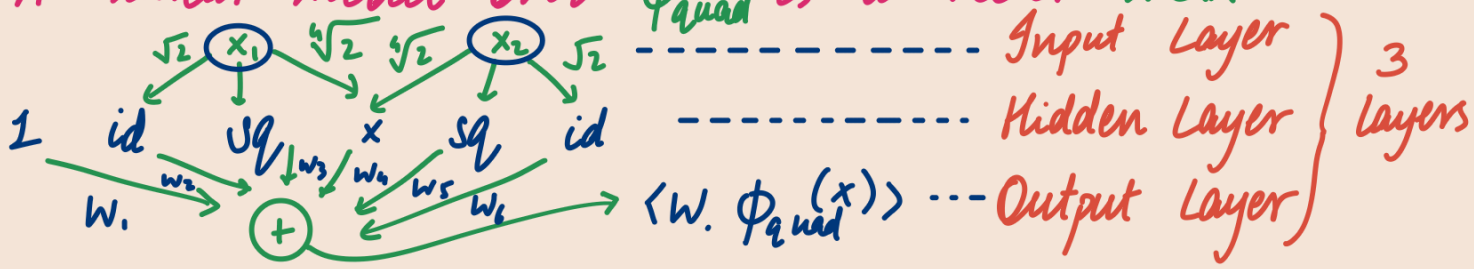
Some drawbacks include: non-convex problem, over parameterized, bulky / slow

KL viewed as DL

- Consider $K_{quad} = (\langle x, y \rangle + 1)^2$ on $\mathcal{X} = \mathbb{R}^2$

$\phi_{quad}((x_1, x_2) = x) = [1, \sqrt{2}x_1, x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2}x_2] \in \mathbb{R}^6$

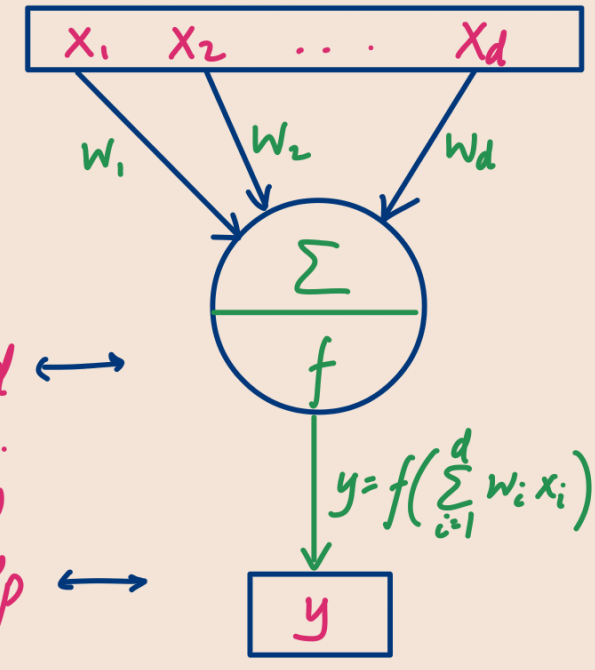
- A linear model over ϕ_{quad} is a vector $w \in \mathbb{R}^6$



- Training kernel RR tunes these weights. I/O layers are called 'visible'.

Neuron in Deep / Neural Networks

- Some inputs can be bias terms (constant) eg. 1
- Input: Input layer: no i/p ↔
  Hidden / Output Layer: i/p taken from other neurons
- Activation: Input Layer: always id ↔
  Output Layer: id / softmax / sigm, etc.
  Hidden Layer: Non-linear eg. ReLU
- Output: Output Layer: final o/p ↔
  Input / hidden layer: o/p to other neurons



- A network w/ no hidden layer is just a generalized linear model, also called a perceptron.

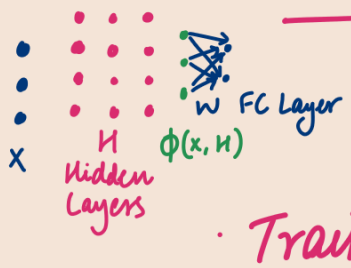
Common Activation Functions

- Sigmoid $\sigma(t) = \frac{e^t}{e^t+1}$; Tan Hyperbolic $\tanh(x) = \frac{e^{2t}-1}{e^{2t}+1} = 2\sigma(2t)-1$
- Rectified (ReLU) $r(t) = [t]_+ = \max(t, 0)$; Leaky ReLU $r(t) = \max(\beta t, t)$ $\beta \in [0, 1]$
  Linear Unit
- Others: Smoothed (exponential, softplus) ReLU, maxout.
- Sigmoid / tanh have a problem of vanishing gradients. (Leaky) ReLU does not face this problem & always learns piecewise linear functions
- Usually, o/p layer neurons: id for regression, sigm for binary / multilabel classification, softmax for multiclassification problems.

Feedforward Deep Networks

- No reverse links allowed, cannot skip layers in classical FF networks.
- Architecture often called a Multi-layered perceptron.
- Generally, same activation function f used from layer to layer. $f(W^T f(U^T f(V^T x)))$
- Generally, the connections are "dense" — all pairs b/w layers are connected.

- The layered architecture is what makes deep networks "deep".
  Lower layers — Computing useful features
  SPN — Learning polynomial etc. functions
  Last layer — Learn a good linear model over these useful features
- A network having a linear activation function in hidden layers is useless. Non-linearity is essential.
- Strictly speaking, NN are parameterized models but, they are frequently over parameterized — millions of parameters.
- # of hidden layers, nodes treated as hyperparameters
- # of output nodes: regression — one node, no activation, binary classification — one node w/ sigmoid, multi-classification — C nodes w/ softmax activation, multi-label-classification — L nodes each w/ sigmoid
- → Final output of NN like $W^T \phi(x, H)$



Loss function

$$\mathcal{L}(W, H) = \frac{1}{n} \cdot \sum_{i=1}^{n} \ell(y_i, W^T \phi(x^i, H))$$

- Training done using mini-batch SGD
- Careful application of chain rule — Backprop

- Step length chosen using Adam, NAG, etc.
- Regularization done explicitly using regularizers and implicitly using dropout.

Adaptive Learning Rates

- Adaptive momentum-based methods popular
- GD is known to experience oscillations.

Idea: Introduce a "velocity" term to prevent changing dir^{n}s abruptly: $v^t = \gamma \cdot v^{t-1} + \eta \cdot \nabla \mathcal{L}(\theta^t)$, $\theta^{t+1} \leftarrow \theta^t - v^t$

- Nesterov's accelerated gradient (NAG): Pioneer method in momentum methods — ϵ convergence in $O(\sqrt{1/\epsilon})$ steps
- Adagrad: Inspiration from Newton's method: $O(1/\epsilon)$ for GD

$\theta^{t+1} \leftarrow \theta^t - (K^t)^{-1} \nabla \mathcal{L}(\theta^t)$; $K^t = \text{diag}(k_1^t \dots k_d^t)$ where $k_i^t = \sqrt{\epsilon + \sum_{\tau=1}^{t} (g_i^\tau)^2}$ → Co-ordinate got rigorous update?

Prevent /0 error ↳ Mellow its future updates

- Reduces to $\eta_t \approx \eta/t$ if all co-ords. got roughly similar gradients in the past $\because k_i^t \approx O(t)$ $\forall i \in [d]$
- RMS Prop: Apply momentum idea to Adagrad.

$k_i^t = \sqrt{\epsilon + v_i^t}$ where $v_i^t = \gamma \cdot v_i^{t-1} + (1-\gamma) \cdot (g_i^t)^2$

Avoids forcing step sizes to go down too much.

- Adam: Combine NAG & RMS-Prop

$g^t = (K^t)^{-1} u^t$ where $K^t = \text{diag}(k_1^t, k_2^t \dots k_d^t)$, $k_i^t = \sqrt{v_i^t + \epsilon}$, & $u^t = \gamma_1 \cdot u^{t-1} + (1-\gamma_1) \cdot g^t$, $v_i^t = \gamma_2 \cdot v_i^{t-1} + (1-\gamma_2)(g_i^t)^2$

Also does some bias corrections on K^t & u^t

How to prevent overfitting?

M1: Regularize the weights

1.1: Add explicit regularization term $\operatorname{argmin}_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta) + \lambda \cdot \|\theta\|_2^2$

1.2: Don't allow any weight to get big $\operatorname{argmin}_{\|\theta\|_\infty < r} \mathcal{L}(\theta)$

M2: Deliberately inject noise into the labels.
Binary classifications: $y^i=0 \rightarrow y^i=\epsilon$, $y^i=1 \rightarrow y^i=1-\epsilon$
For regression: $y^i \rightarrow y^i+\epsilon$ where $\epsilon \sim N(0,\sigma^2)$
· Heuristics for NN, but can be shown to be equivalent to regularization for others. Unlikely, that an NN can learn noise w/ a limited # of nodes.

M3: Learn Sparse Models
3.1: Sparse connections b/w layers instead of dense.
3.2: Parameter sharing: Force some of the weights to take the same value by adding constraints. CNN do this & are very successful.
3.3: Dropout: Implicitly train on multiple sparse networks in parallel. Randomly remove edges / nodes to not update them in an iteration. Put them back after the update.

M4: Validation: Use early stopping not on training loss but performance on held-out data.

Dropout at test time: Scale the (post-activation) output of each node in the NN w/ the prob. w/ which that would have been spared from marking
· Forces nodes to learn to work in the absence of other nodes — robust.
$P[drop] = 0.2/0.5$ for i/p / hidden layers usually.

Popular layer types: Perceptron, Residual / Skip Layer, Convolution layer, Transformer Layer.

Popular Training Strategies

Pretraining: Entire network composed of $\theta := (w, H)$.
· Get H from some other source

- During train, freeze H and only train w — very fast!
- Can also train H w/ w — known as "fine-tuning" of H. Train only the top few layers of H for speed. Can also train all layers of H if pretraining task was very different.

Siamese: (Two-tower models)
- Denote $d(i,j,H) := \|\phi(x^i,H) - \phi(x^j,H)\|_2$.
- Given a C-class problem, encourage $d(i,j,H) \to 0$ if $y^i = y^j$ else want $d(i,j,H) \gg 0$
- Let (i,j,k) be a triplet s.t. $y^i = y^j \neq y^k$
- Triplet loss function: $[d(i,j,H) - d(i,k,H) + \lambda]_+$ (Δ should be λ at least)
- Contrastive loss function: $[\underline{d(i,j,H)} - \lambda]_+ + [\lambda - \underline{d(i,k,H)}]_+$
- Popular in vision, NLP, recomm. (should not be $> \lambda$; shouldn't be $< \lambda$)

Cloze Tasks: Very popular in training language models, graph models. Given an object, hide it and ask the NN to retrieve it.
- Also known as masked language modelling in NLP.
- Train network parameters H so that:
  $\|\phi(\langle MASK \rangle, H) - \phi(batman, H)\|_2 \to 0$
  Training often proceeds in Siamese style.
- To train graph models, can train the network to predict neighbours.

Recommendation Systems (RecSys)

- There are users, and there are items. Users come to us one at a time in no particular order and we need to recommend to each user, items they will like. User tastes expected to be stable but may

vary slowly across time.
- Ex: (Social Media user, Posts), (Student, Courses), (Patient, Medicines), (Search query, Ads), (Advertiser, Search query to bid on)

RecSys via Matrix Completion: m users, n items
- Rating matrix $X_{m \times n}$, X_{ij}: rating of how much user i likes item j
- Some users rate some items
  ↳ Very rarely → Get to see those entries of X
  Can we recover the unseen entries, ie, Complete X
  ↳ Would reveal the most liked item of each user.
- We don't assume we have access to any user/item features in this model (instead learn them by modelling them as latent variables.
- Some assumptions → Necessary to solve the problem
  users similar ⇒ rate all items similarly
  items similar ⇒ all users rate them similarly
- Assume for every user u_i, exists a vector $u^i \in \mathbb{R}^k$ & for every item v_j, exists a vector $v^j \in \mathbb{R}^k$ s.t. rating $r_{ij} = \langle u^i, v^j \rangle + \epsilon_{ij}$, $\epsilon_{ij} \sim N(0, \sigma^2)$. Equivalent to saying that the ratings matrix is almost rank k
- $U = [u^1 \dots u^m]^T$, $V = [v^1 \dots v^n]^T$ then $X = UV^T + E$, k is a hyperparameter.
- If two users rate similarly, then the method will learn similar "features" for them. These can be useful for future recomms. But can also violate privacy if they reveal details users didn't tell us, eg, correlation w/ age.
- Suppose $\Omega \subset [m] \times [n]$ are the locations we have observed. The MAP problem (w/ Gaussian priors on

u^i, v^j reduces to the following optimization problem:

$$\underset{\substack{U \in \mathbb{R}^{m \times k} \\ V \in \mathbb{R}^{n \times k}}}{\operatorname{argmin}} \sum_{(i,j) \in \Omega} (X_{ij} - \langle u^i, v^j \rangle)^2 + \lambda \cdot \left(\sum_{i=1}^{m} \|u^i\|_2^2 + \sum_{j=1}^{n} \|v^j\|_2^2 \right)$$

AltMin for Low-Rank Matrix Completion

Alternating Optimization: 1. Ω & values $\{X_{ij} : (i,j) \in \Omega\}$

2. Init $v^{1,0}, v^{2,0}, \ldots, v^{n,0}$ 3. For $t = 1 \ldots$

3.1. $u^i \leftarrow \left(\sum_{j:(i,j) \in \Omega} v^j (v^j)^T + \lambda I \right)^{-1} \left(\sum_{j:(i,j) \in \Omega} X_{ij} \cdot v^j \right) \quad \forall\ i = 1 \ldots m$

3.2. $v^j \leftarrow \left(\sum_{i:(i,j) \in \Omega} u^i (u^i)^T + \lambda I \right)^{-1} \left(\sum_{i:(i,j) \in \Omega} X_{ij} \cdot u^i \right) \quad \forall\ j = 1 \ldots n$

4. Repeat until convergence

- Can speed up by using SGD to update user and item latent vectors — $O(k)$/iteration
- Suppose ratings are not $\mathbb{R}$ but $\mathbb{N}$ or Boolean
  - Can model them using generalized linear models instead — $\mathbb{P}[r_{ij} | u^i, v^j] \propto \exp(r_{ij} \langle u^i, v^j \rangle - A(\langle u^i, v^j \rangle) - h(r_{ij}))$
  - The model studied above ⇝ special case → $\mathcal{N}(\langle u^i, v^j \rangle, \sigma^2)$
  - Instead of the alternations being ridge regression problems, they would now become GLM (eg. log. reg. for bools)
- ∃ non-linear extensions to these as well

Pros: · Doesn't require personal/item info.
· Collaborative method: Lazy users benefit from active users

Cons: · Can't utilize user/item info even if given
· Expensive to recompute matrix factorization again & again
· Adding new users & items not straightforward.

RecSys via Multi-label Learning

- Every user represented as a vector $x \in \mathbb{R}^d$.
- L items — every item $j \in [L]$ is a potential label.

· Training data gives us $\{(x^i, y^i)\}_{i=1}^n$, $y^i \in \{(0,1)\}^L$
Learn DT / LwP / deep models to predict y^i using x^i

* $y_j^i = 1 \Rightarrow$ User i does like item j
$= 0 \Rightarrow$ Does not necessarily mean i doesn't like j
$\because$ users mostly do not tell us anything about items they don't like
Influences what performance measures can be used.

Mostly precision based performance measures used $\because$ recall based performance measures require us to know the entire set of liked items — unreasonable to expect

RecSys using kNN

· Find neighbours using the feature vectors of users (x^i), eg. Euclidean / Mahalanobis distance.
· Once neighbours known, find out which labels are popular in the neighbourhood and recommend top # labels.
· May similarly use LwP methods too. Average the feature vectors of all users who like a label as the "prototype" of that label — total of L prototypes.
At test time, recommend whichever labels are s.t. their prototypes are close to the test feature vector.

* LwP/kNN may give better performance for rare labels

RecSys using DT: Instead of just storing a few labels at leaves, can store how popular those labels are as well (eg. count how many times each active label occurred in train points that reached that leaf) — can allow the labels to be ranked.

RecSys using Linear Models

- Treat each label as independent binary classification problem — approach called **binary relevance**
- Good accuracies but not scalable if L is large.
- Total of L binary problems — learn a separate linear model for each eg. $y_i^i = \text{sign}(\langle w^j, x^i \rangle)$
  - Probabilistic methods ← logistic regression
  - → SVM

RecSys using Latent Variables

- Part of the explosion in number of users / items may be an illusion.
  - 1 M users/items ⇝ May not be 1M types of users/items
  - There may very well exist a small set of item types s.t. prediction of any item ≡ combination of these types
- Simple idea: Force the predictors $w^j \in \mathbb{R}^d$ to be LCs of a few, say k meta predictors, say $z^1 \dots z^k \in \mathbb{R}^d$
  - Have to learn both $z^1 \dots z^k$ & how to combine them to get w^j, ie, $h^j \in \mathbb{R}^k$ s.t. $w^j = h_1^j \cdot z^1 + \dots + h_k^j z^k$
  - ↪ Equivalent to assuming $\text{rank}(W = [w^1 \dots w^L] \in \mathbb{R}^{d \times L}) = k$
- There may ∃ a small set of item combinations (say those that are frequently purchased together) that every purchase profile can be explained in their terms.
- Simple idea: Assume that the like prob. vector of every user, ie, $\eta \in [0,1]^L$ is an LC of a few prototype prob. vectors — each prototype corresponding to a purchase profile.

- Pros:
  - Very powerful — can extend to millions of items and users.
  - SoTA accuracies, Very fast training + predictions

- Can utilize user / item features & add new users easily

- Cons:
  - No explicit collaboration — does not infer similarity of items / users. May pose an issue if user features aren't very informative
  - Not straightforward to add new items
  - Recommendation can get stale with time.

Fin