Support Vector Machine

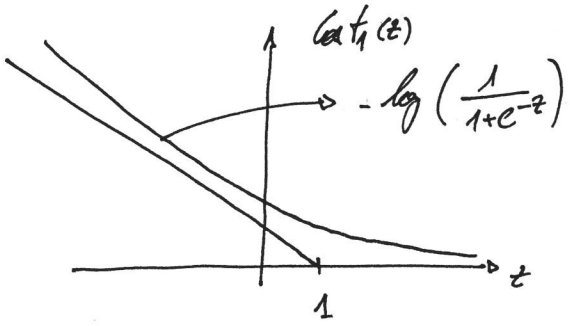
* Alternative to logistic regression → Non-linear solutions.

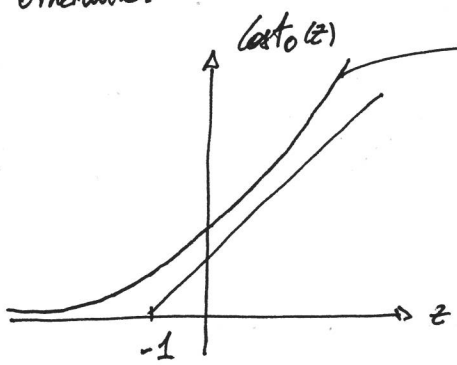
Objective → $\min_\theta \frac{-1}{m}\left[\sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\,\text{cost}_2(\theta^T x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

gets rid off constant, doesn't impact minimization.

$$\text{Cost} = \min_\theta \; C\sum_{i=1}^{m}\left[y^{(i)}\,\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\,\text{cost}_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

Hypothesis → $h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{Otherwise.} \end{cases}$





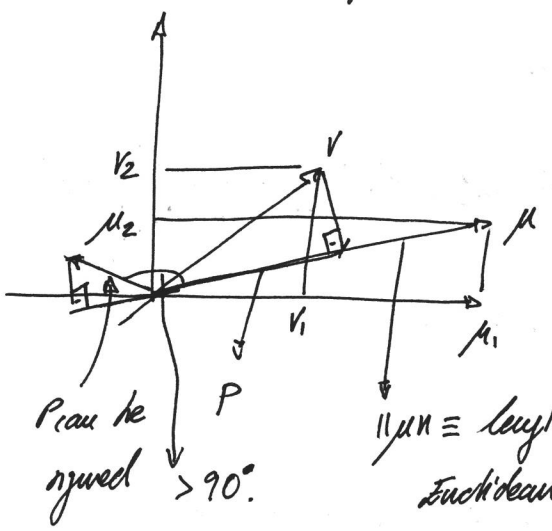
* to to reduce cost → If $y=1$ → $\theta^T x \geq 1$ → $\text{cost}_1(z) = 0$.

If $y=0$ → $\theta^T x \leq -1$ → $\text{cost}_0(z) = 0$.

* Theory behind $C = \frac{1}{\lambda}$; trade off between error cost and regularization.

* If C ↑↑↑, it will cause overfit.

+ Vector inner product ⟶



$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} ; \quad V = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$$

$p \equiv$ length of the projection of vector v on μ.

$p \in \mathbb{R}$

$p' \equiv$ length of the projection of vector μ on v.

$\|\mu\| \equiv$ length of vector μ

Euclidean distance.

$$\|\mu\| = \sqrt{\mu_1^2 + \mu_2^2} \in \mathbb{R}$$

$$\mu^T \cdot v = v^T \cdot \mu = p \cdot \|\mu\| = p' \|v\| = v_1 \mu_1 + v_2 \mu_2$$

+ SVM decision boundary ⟶

$$\min_\theta \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

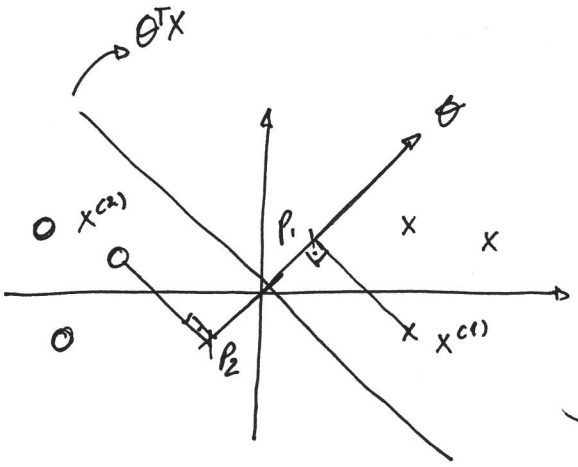
s.t $\theta^T x \geq 1$ If $y = 1$

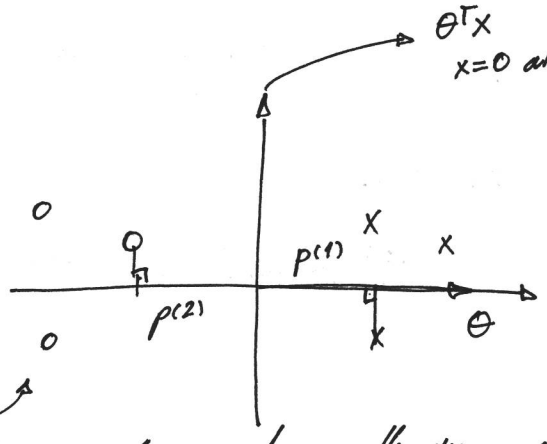
s.t $\theta^T x \leq -1$ If $y = 0$.

simplifying ⟶ $\theta_0 = 0$; $n = 2$

$$\min_\theta \frac{1}{2}(\theta_1^2 + \theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2 + \theta_2^2}\right)^2 = \frac{1}{2}\|\theta\|^2$$ ⟵ minimizing this of cost$_1$(z) already completed.

$$\theta^T x^{(i)} = p^{(i)} \|\theta\| = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$$





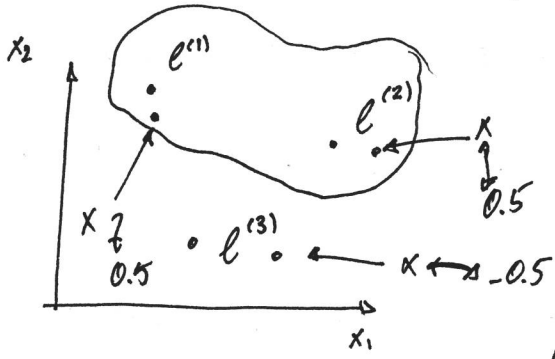
$p^{(i)}$ take larger values, allowing a smaller $\|\theta\|$

* Kernels ⟶

$$z = \theta^T f = \theta_0 + \theta_1 f_1 + \theta_2 * f_2 + \dots + f_m \theta_m$$

$$f_i^{(i)} = \text{similarity}(x^{(i)}, l^{(i)}) \; ; \quad f \in \mathbb{R}^{m+1} \; ; \quad \theta \in \mathbb{R}^{m+1} \; ; \quad f = \begin{bmatrix} f_0^{(i)} = 1 \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$$

- Landmarks are choosen as every input example ⟶ m samples.



Among 2 diferences.

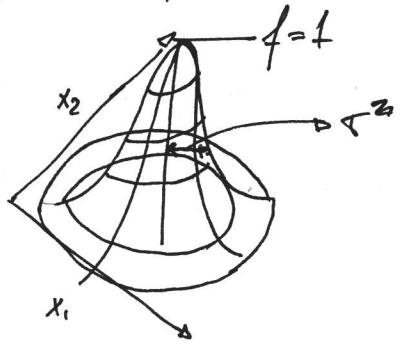
$\theta_0 = -0.5, \; \theta_1 = 1 \; ; \; \theta_2 = 1 \; ; \; \theta_3 = 0$

$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0 \longrightarrow \hat{y} = 1.$

Given this θ values, any point close to $l^{(1)}$, $l^{(2)}$ ⟶ $\hat{y} = 1$. Other, $\hat{y} = 0$.

- Gaussian Kernel ⟶

$$f = \text{similarity}(x^{(i)}, l^{(j)}) = \exp\left(-\frac{\| x^{(i)} - l^{(j)} \|^2}{2\sigma^2}\right)$$



$\sigma^2 \equiv$ Control how sharp the function is, so for small σ less penalty to the difference.

Change dim.

- Cost function ⟶

$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} \, \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \, \text{cost}_0(\theta^T f^{(i)}) \right] + \sum_{i=1}^{m} \frac{\theta_i^2}{2}$$

* In reality →

$\theta_j = -\theta^T \cdot \theta \longrightarrow -\theta^T \cdot M \cdot \theta$

If we have large data set, really expense to compute. $\underline{m = 10^6}$

* Choosing parameters →

$C \left(= \frac{1}{\lambda}\right) \longrightarrow$ Large $C \equiv$ Lower bias, high variance.

Small $C \equiv$ High bias, low variance. } overfit.

$\sigma^2 \longrightarrow$ Large $\sigma^2 \equiv$ Higher bias, low variance

Small $\sigma^2 \equiv$ Low bias, high variance

- In practice →

1. Need to choose C
2. Need to choose kernel → Not all fuctions works, comply "Mercel Theorem". to they don't Diverge

A. No Kernel; "Linear kernel" → $y = 1$ if $\theta^T x \geq 0$.

n large and small m → avoid overfitting.

B. Gaussian →

Need to choose σ^2?

n small and/or large m.

Important to do feature scaling. → Imports $\|x - \ell\|^2$

* Other Kernels →

- Polynomial Kernel →

$K(x, l) = (x^T l + \text{const})^{\text{degree}}$

* Usually need when x, l non negative.

* Need to decide const. degree.

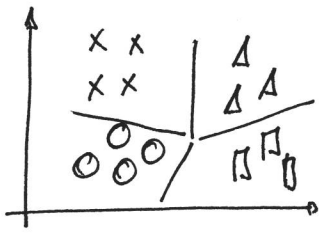
- String, chi-square, histogram intersection.

* Multiclass classification →

- Use one-vs-all method

- Train K SVM, one to distinguish $y = i$ from the rest $i = 1, 2, \dots, K$

$\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)}$ → $y = 1, y = 2, \dots, y = K$.



* Logistic regression vs SVM →

- n is large relative to m → logistic regression or SVM linear kernel.

- n is small (1-1000), m (10-$10{,}000$) →

SVM gaussian Kernel

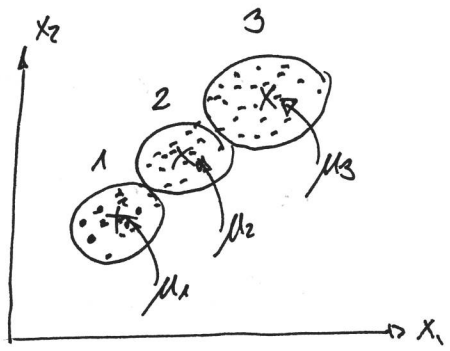
- n is small, m is large →

$n = 1$-1000
$m = 50{,}000+$

→ 1. Create new features.

2. Logistic regression or SVM linear Kernel.

K-Means Clustering



* Unsupervised learning
* Algorithm that groups point by itself.
* Implementation →

Inputs →

$K \equiv$ Number of clusters.

Training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

1. Randomly initialize K cluster centroids $\mu_1, \mu_2, \mu_3, \ldots, \mu_K$.

2. Repeat {

for $i = 1 : m$,

Cluster Assignment. → $c^{(i)} :=$ index (from 1 to K) of cluster centroid to which $x^{(i)}$ is assigned → $\min_K \| x^{(i)} - \mu \|^2$

Move centroids. →

for $K = 1 : K$,

$\mu_K :=$ average (mean) of points assigned to cluster K.

end;

end; ↑

If there's a centroid with no points → delete.

* Optimization →

$c^{(i)}$ ≡ index of cluster $(1, 2, \ldots, K)$ to which example $x^{(i)}$ is assigned.

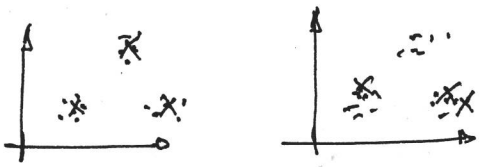
μ_k ≡ Cluster centroid k ($\mu_k \in \mathbb{R}^n$)

$\mu_{c^{(i)}}$ ≡ Cluster centroid to which $x^{(i)}$ has been assigned.

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \mu_{c^{(i)}} \|^2$$

* for each example minimize the distance between the point and its cluster centroid.

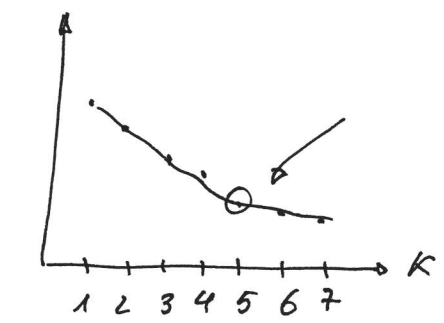
* Random initialization →



- K-Mean may end up in a local optima
- Run several randomization and pick clustering lower cost.
- If K is very large, many multiple random initialization won't make much difference.

* Number of Clusters →

- Normally chosen manually, through data visualization.
- Elbow →



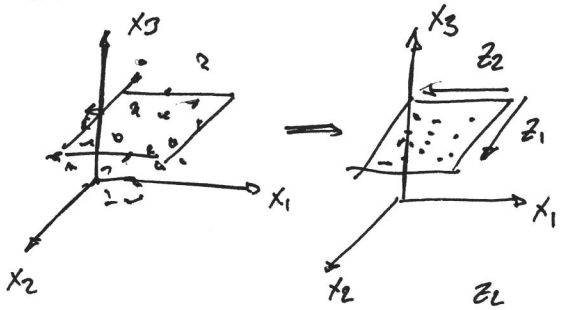
- Based on next metric needs. Elbow is not always clear.

* Data compression & visualization →

Country	GDP	Per capita GDP	Human dev. index	Life expect.	Poverty index
Canada →		→			

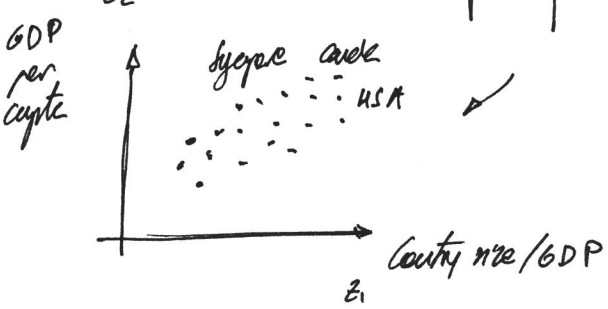
$x^{(i)} \in \mathbb{R}^{50}$

3D →

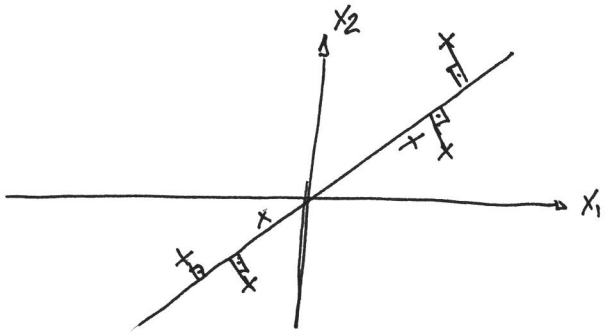


Country	z_1	z_2
Canada	1.6	1.2

$z^{(i)} \in \mathbb{R}^2$

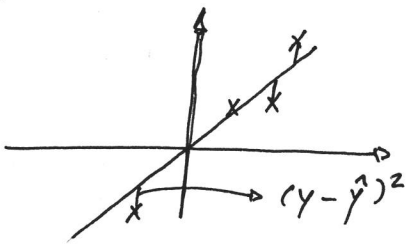


Principal Component Analysis



$x^{(i)} \in \mathbb{R}^2$

Difference to linear regression



* Finds a lower dimension hyperplane in which the projection has the lower distance to the points.

* Algorithm ⟶

A. Data Normalization ⟶ Each feature.

$$x_j = \frac{x_j - \mu}{\sqrt{\sigma^2}}$$

B. Reduce data from n-dimension to k-dimension ⟶

$$x^{(i)} \longrightarrow z^{(i)} \in \mathbb{R}^k$$

1. Compute "covariance matrix" ⟶

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} \underbrace{x^{(i)} \cdot x^{(i)T}}_{\underbrace{n \times 1 \quad 1 \times n}_{n \times n}} \longrightarrow \text{sigma} = \frac{1}{m} X' \cdot X$$

2. Compute "eigenvectors" of matrix Σ ⟶

"singular value decomposition"

$$[U, S, V] = \text{svd}(\text{sigma}) \longrightarrow n \times n$$

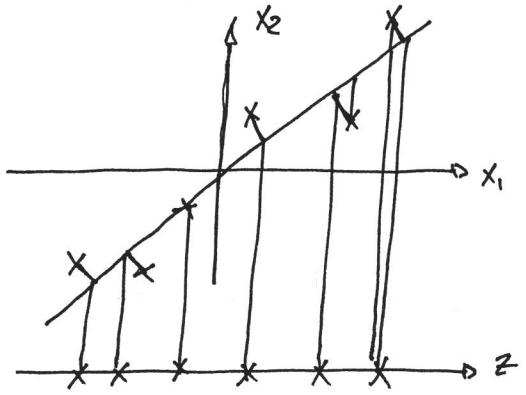
$U = [\ u^{(1)}\ u^{(2)} \longrightarrow u^{(n)}\ ]$

K vectors from matrix

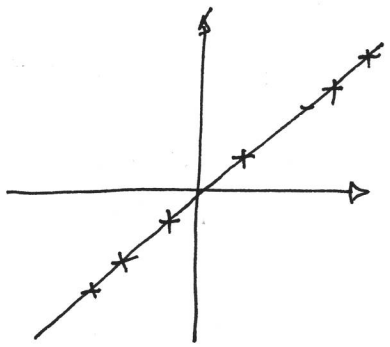
$U_{reduce} \equiv \dim(n, k) \equiv U(:, 1:K)$

$z = U_{reduce}^{T} \cdot X \longrightarrow \dim(m, k)^{T} \cdot \dim(n \times 1) = \dim(K, 1)$

* Reconstruction from compressed data →



$z = U_{reduce}^{T} \cdot X$



$x_{approx} = U_{reduce} \cdot z$

* Selecting # PCA →

- Objective → minimize average square error projection.

$$Error = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - x^{(i)}_{approx} \|^2$$

- Total variation of my data →

On average how far are my samples from 0.

$$1/m \sum_{i=1}^{m} \| x^{(i)} \|^2$$

* Choose k to be the smallest in which we retain a selected variance ⟶

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x^{(i)}_{approx} \|^2}{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01 \ (1\%) \longrightarrow \text{99\% of variance retained.}$$

* Algorithm ⟶

- Iterate through k values ⟶

  * Compute U_{reduce} : $z^{(1)} \longrightarrow z^{(m)}$

  * Calculate $x^{(1)}_{approx}, \ldots, x^{(m)}_{approx}$.

  * Check variance retained.

- k values $(1, 2, \ldots, 17)$; 17 lady kept it ⟶ $k = 17$.

- How to calculate in Octave ⟶

```
[U, S, V] = svd(sigma)
```

$$S = \begin{bmatrix} S_{11} & 0 & \cdots & 0 \\ 0 & S_{12} & S_{13} & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & S_{nn} \end{bmatrix}$$

k ↙ n ↙

For a given k ⟶

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01$$

* PCA applications ⟶

1. Speed-up supervised learning ⟶

   * Dimensionality reduction.

   $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)}) \in \mathbb{R}^{10000}$

   PCA ↓

   $(z^{(1)}, y^{(1)}), \ldots, (z^{(m)}, y^{(m)}) \in \mathbb{R}^{1000}$

   * Mapping should only be applied by the training, validation or test sets.

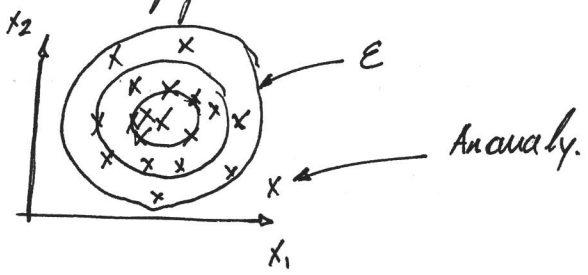
2. Data visualization ⟶ $k=2$, $k=3$.

3. Bad use ⟶ Prevent overfitting.

   * Using this dimensionality reduction may throw an looking relationships between directions reduced and labels.

Anomaly Detection

simplification



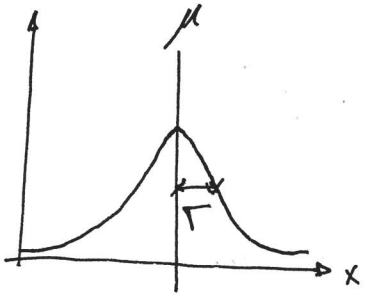
Dataset = $\{ x^{(1)}, x^{(2)}, \ldots, x^{(m)} \}$

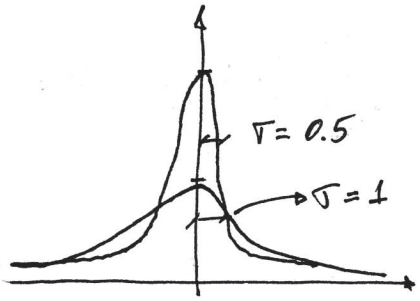
Model $p(x)$; $x = \{ x_1, x_2, \ldots, x_n \}$

$\longrightarrow$ $p(x_{test}) < \varepsilon \longrightarrow$ Anomaly.

$p(x_{test}) \geq \varepsilon \longrightarrow$ OK.

* Gaussian distribution $\longrightarrow$





Area of $p(x; \mu, \sigma^2) = 1$.

Probability distribution.

$x \sim N(\mu, \sigma^2)$

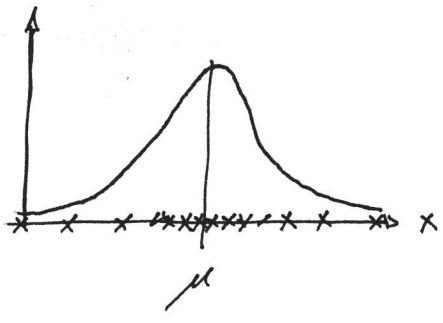
$$p(x) = p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Parameter estimation $\longrightarrow$

Dataset = $\{ x^{(1)}, x^{(2)}, \ldots, x^{(m)} \}$; $x^{(i)} \in \mathbb{R}$.

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

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



Maximum likelihood estimates.

* Algorithm ⟶

Training set ⟶ $\{x^{(1)}, \ldots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$

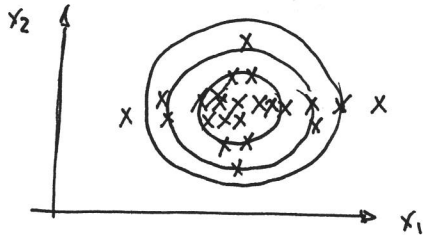
$$p(x) = p(x_1; \mu_1, \sigma_1^2) \cdot p(x_2; \mu_2, \sigma_2^2) \cdot \ldots \cdot p(x_n; \mu_n, \sigma_n^2)$$

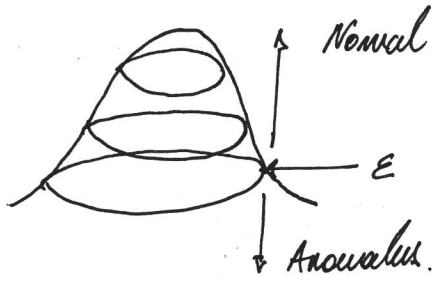
Independent assumption.

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

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

1. Choose features that might indicate anomalous examples.
2. Fit parameters ⟶ $\mu_1, \ldots, \mu_j, \sigma_j^2$
3. Create model ⟶ $p(x)$
4. Anomaly if $p(x) < \varepsilon$.





- Algorithm evaluation →

  * Have labeled data for evaluation. Eg. →

    - Training set → 6000 good → $p(x)$ model.
    - Cross-validation → 2000 good / 10 anomalies.
    - Test → 2000 good / 10 anomalies.

  * Flow →

    1. Fit model $p(x)$ on training set → $\{x^{(1)}, \ldots, x^{(m)}\}$
    2. On cross-validation / test example x →

$$y = 1 \text{ if } p(x) \leq \varepsilon \text{ Anomaly}$$

$$y = 0 \text{ if } p(x) > \varepsilon \text{ Normal.}$$

    3. Possible evaluation metrics → Very skewed data.
       * True positive, false positive, false negative, false positive.
       * Precision / Recall.
       * F_1-score → Learning summary of precision / recall.

    4. Optional → You can use cross-validation to set parameter ε.

- When to use instead of Supervised learning →

  * Very small number of positive examples, and large negative.
  * Many different types of anomalies
  * Future anomalies may look nothing like previous.

- Choosing features ⟶

  * Features are modeled as gaussians
  * If features don't have this distribution. Transformations ⟶
    * $\log(x + c)$
    * x^c
  * Anomaly not captured by current features ⟶ Create new feature.

Eg: All features ⟶ $x_1 \equiv$ memory use of computer.

$x_2 \equiv$ number of disk accesses/sec

$x_3 \equiv$ CPU load.

$x_4 \equiv$ Network traffic

New feature ⟶ $x_5 = \frac{\text{CPU load}}{\text{Network traffic}}$ or $\frac{\text{CPU load}^2}{\text{Network traffic}}$.

This will capture anomalies of computer stuck in inf. loops.

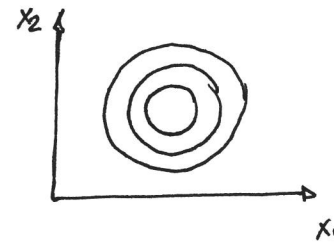
- Multivariate Gaussian Distribution ⟶

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}\left((x-\mu)^T \Sigma^{-1} (x-\mu)\right)}$$

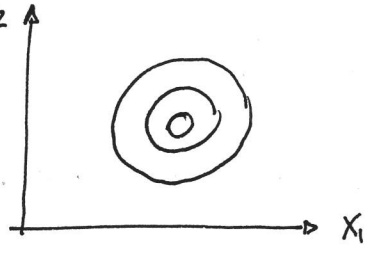
$\mu \in \mathbb{R}^n$

$\Sigma \in \mathbb{R}^{n \times n}$ (Covariance Matrix). ; $|\Sigma| \equiv$ Determinant of Covariance.

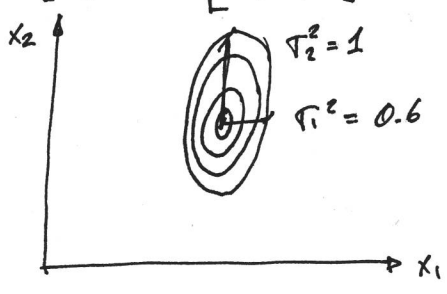
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



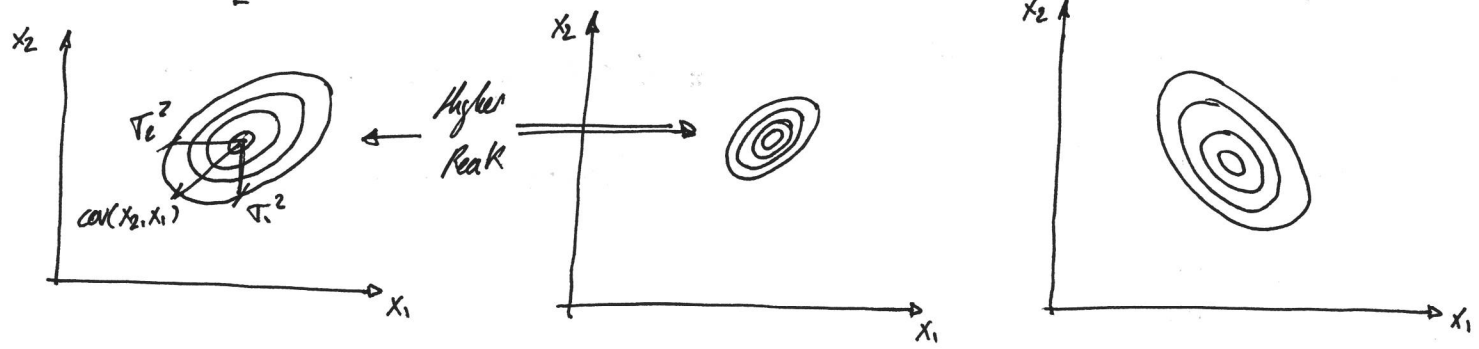
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$



$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$



$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$



* Differently from covariance with itself, covariance between ~~values~~ peakes while variables increasing.

* Application ⟶

Training set = $\{ x^{(1)}, x^{(2)}, \dots, x^{(m)} \}$

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

- Given example ⟶ $p(x_{test}) < \varepsilon$?

* Original model is a special case of the multivariate model ⟶

It assumes that variables are independent.

$$\Sigma = \text{diag matrix} = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_n^2 \end{bmatrix}$$

Original Model

* Manually create features to highlight anomalies

$$x_3 = \frac{x_1}{x_2}$$

* Computationally cheaper

* Work ok with small number of samples m.

Multivariate Model

* Automatically captures correlation between features.

* More expensive $\rightarrow$

$$\Sigma \in \mathbb{R}^{n \times n} ; \Sigma^{-1}$$

* Must have $m > n$ or else

Σ is not invertible

Σ singular matrix.

- Features that are not linearly independent $\rightarrow$ $x_3 = x_4 + x_5$

Σ^{-1} not invertible.

Recommender systems →

* Content Based Recommendation →

Movies	Person 1	Person 2	Person 3	Person 4	Love Features. x_1 (Romance)	x_2 (action)
Movie 1	5	5	0	0	0.9	0
Movie 2	5	?	?	0	1.0	0.01
Movie 3	?	4	0	?	0.99	0
Movie 4	0	0	5	4	0.1	1.0
Movie 5	0	0	5	?	0	0.9

- Nomenclature →

* $n_u \equiv$ # users. → $n_u = 4$; $n_m = 5$.
* $n_m \equiv$ # movies.
* $r(i,j) = 1$ if user j has reviewed movie i.
* $y(i,j)$ = Rating given by user j to movie i.
  Defined only if $r(i,j) = 1$.
* $\theta^{(j)} \equiv$ Parameter vector for user j.
* $x^{(i)} \equiv$ Parameter vector for movie i. Features.
* Prediction → $(\theta^{(j)})^T (x^{(i)})$
* $m^{(j)} \equiv$ # of movies rated by user j.

- Cost function →

$$J(\theta^{(1)}, \dots, \theta^{(n_u)}) = \min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i : r(i,j)=1} \left(\theta^{(j)T} x^{(i)} - y^{(i,j)}\right)^2$$

$$+ \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

Remove $m^{(j)}$ because a constant factor doesn't impact the minimization rule for $\theta^{(j)}$

- Gradient descent update →

$$\theta_k^{(j)} \leftarrow \theta_k^{(j)} - \alpha \sum_{i : r(i,j)=1} \left(\theta^{(j)T} \cdot x^{(i)} - y^{(i,j)}\right) \cdot x_k^{(i)} \quad \text{for } k = 0.$$

$$\theta_k^{(j)} \leftarrow \theta_k^{(j)} - \alpha \left[\sum_{i : r(i,j)=1} \left(\theta^{(j)T} \cdot x^{(i)} - y^{(i,j)}\right) \cdot x_k^{(i)} + \lambda \theta_k^{(j)}\right] \quad \text{for } k \neq 0.$$

* Collaborative Filtering →

- In this case, no knowledge of the features →

Movie	Person 1	Person 2	Person 3	Person 4	x_1 (romance)	x_2 (action)
Movie 1	5	5	0	0	?	?
Movie 2	5	?	?	0	?	?
Movie 3	2	4	0	?	?	?
Movie 4	0	0	5	4	?	?

- If we know $\theta^{(j)}$ →

$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ → $(\theta^{(1)})^T x^{(1)} \approx 5$ → We can infer features.

- Cost function →

$$J(x^{(1)}, \dots, x^{(n_m)}) = \min_{x^{(i)}} \frac{1}{2} \sum_{\substack{j: r(i,j)=1 \\ \left(\sum_{i=1}^{n_m}\right.}} (\theta^{(j)T} x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

- Gradient descent update →

$$x_k^{(i)} \leftarrow x_k^{(i)} - \alpha \left(\sum_{j: r(i,j)=1} (\theta^{(j)T} x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

* Recommender Algorithm →

- Cost function →

$$J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j): r(i,j)=1} (\theta^{(j)T} x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{1}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \theta_k^{(j)2}$$

* $i: r(i,j)=1$ → For every movie rated by user.

* $j: r(i,j)=1$ → For every user that rated movie i.

$x_0 = \theta_0 = 0$

$x \in \mathbb{R}^n, \theta \in \mathbb{R}^n$

- Algorithm →

1. Initialize $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ random variables.

2. Minimize $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$

$$x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j: r(i,j)=1} (\theta^{(j)T} x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i: r(i,j)=1} (\theta^{(j)T} x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

3. For user with parameters θ and movie with learned features x, predict a rating $\theta^T x$.

- Low Rank Matrix Factorization ⟶

* Collaborative filtering ⟶

Movie	Person 1	Person 2	Person 3	Person 4
Movie 1	5	5	0	0
Movie 2	5	?	?	0
Movie 3	?	4	0	?
Movie 4	0	0	5	4
Movie 5	0	0	5	?

↙

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

$n_m \equiv$ # movies.

$n_u \equiv$ # users.

$Y \in \mathbb{R}^{(n_m, n_u)}$

$$Y = \begin{bmatrix} \theta^{(1)T} x^{(1)} & \theta^{(2)T} x^{(2)} & \dots & \theta^{(n_u)T} x^{(1)} \\ \vdots & & & \\ \theta^{(1)T} x^{(n_m)} & \theta^{(2)T} x^{(n_m)} & \dots & \theta^{(n_u)T} x^{(n_m)} \end{bmatrix} \in \mathbb{R}^{(n_m, n_u)}$$

$$X = \begin{bmatrix} — & x^{(1)T} & — \\ & \vdots & \\ — & x^{(n_m)T} & — \end{bmatrix} \quad ; \quad \Theta = \begin{bmatrix} — & \theta^{(1)T} & — \\ & \vdots & \\ — & \theta^{(n_u)T} & — \end{bmatrix}$$

; $X \in \mathbb{R}^{(n_m,\ \text{movie features})}$

; $\Theta \in \mathbb{R}^{(n_u,\ \text{movie features})}$

$Y = X \Theta^T$

- Find related movies ⟶ How to find movie j related to movie i?

$\| x^{(i)} - x^{(j)} \|$ ⟶ small.

* Mean normalization ⟶

- User without rated movies.

- Problem ⟶

  * If we run this over the cost function $\theta^{(\text{new user})} = 0$.

  * Since regularization for $\theta^{(j)}$ will keep $\theta^{(j)}$ value low, and there's no defined targets ⟶ $\theta^{(j)} = 0$.

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \longrightarrow \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \longrightarrow Y - \mu$$

- For user j, on movie i predict ⟶

$$\theta^{(j)T} x^{(i)} + \mu_i$$

  * User 5 (Eve) ⟶ New user.

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \longrightarrow \underbrace{(\theta^{(5)})^T x^{(i)}}_{\to 0} + \mu_i$$

Applying Machine Learning.

* Evaluating hypothesis →

Size	Price
↕ → Train set 70%	
↕ → Test set 30%	

Randomize before selecting sets.

- On classification you may use other error definition instead of cross-entropy →

Misclassification error

$$error(h_\theta(x), y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5,\ y = 0 \text{ or} \\ & h_\theta(x) < 0.5,\ y = 1 \\ 0 & \text{Otherwise} \end{cases}$$

$$\text{Test error} = \frac{1}{M_{test}} \sum_{i=1}^{M_{test}} error\left(h_\theta\left(x^{(i)}_{test}\right), y^{(i)}_{test}\right)$$

* Model classification →

Train set 60%

Cross-validation 20%

Test set 20%

$d = 1 \rightarrow h_\theta(x) = \theta_0 + \theta_1 x \rightarrow \theta^{(1)}$

$d = 2 \rightarrow h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

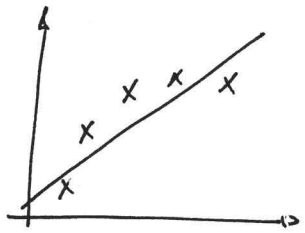
⋮

$d = 10 \rightarrow h_\theta(x) = \theta_0 + \theta_1 x^2 + \dots + \theta_{10} x^{10}$

- Use cross-validation set to select the model →

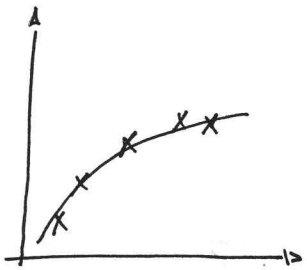
$J_{CV}(\theta^{(1)}), \dots, J_{CV}(\theta^{(10)})$ → use smaller error.

* Diagnosis bias vs variance → Model selection.



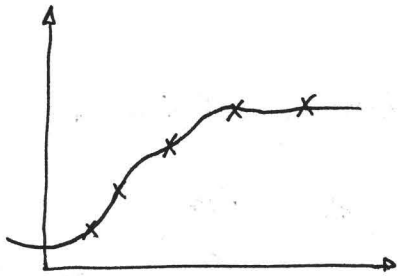
$\theta_0 + \theta_1 x$

High bias (underfit)



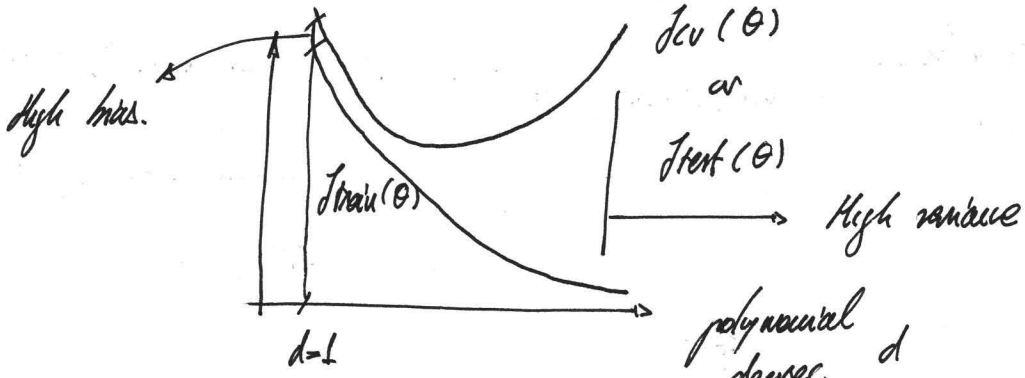
$\theta_0 + \theta_1 x + \theta_2 x^2$

Correct.



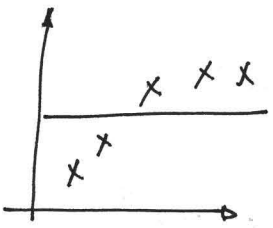
$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance (overfit).

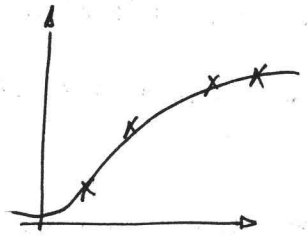


* Regularization and bias / variance →

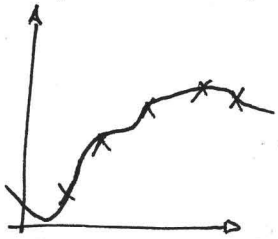
Model → $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$; Training cost → $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$



Large λ
High bias
$\lambda = 10000$
$\theta_0 \approx \theta_1 \approx \theta_2 \approx \ldots \approx 0$

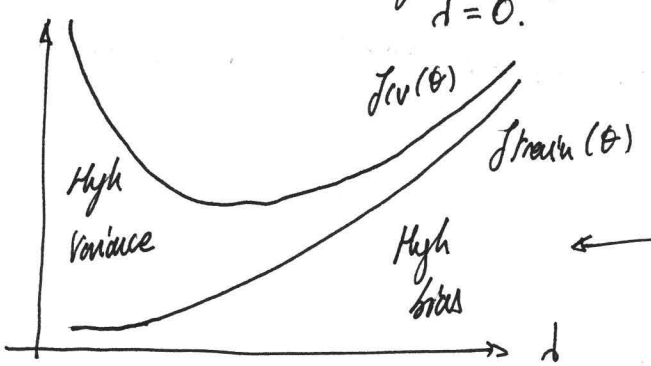


Intermediate λ
Correct



Small λ
High Variance
$\lambda = 0$.

$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$; $J_{cv}(\theta)$; $J_{test}(\theta)$

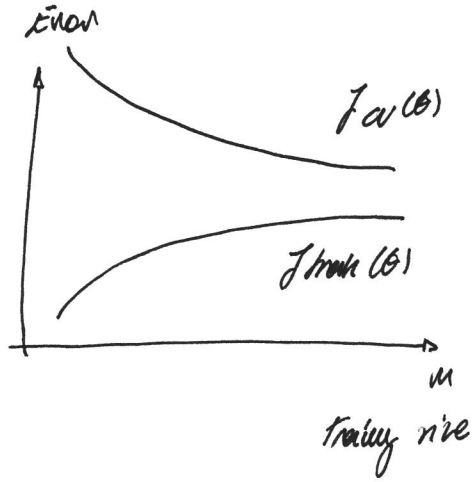


← Select one with lowest cost.

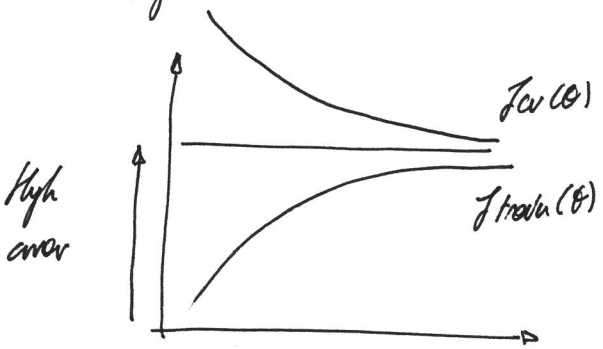
* Learning curves →

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$J_{cv}(\theta) = \frac{1}{2m} \sum_{i=1}^{m_{cv}} \left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2$$

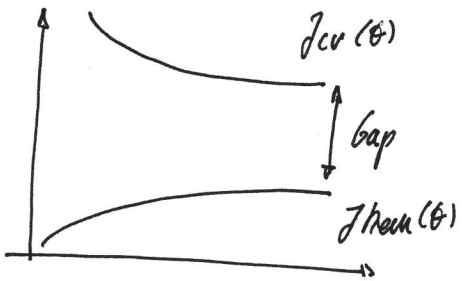


- High bias →



* Simple model
* More data won't help.

- High Variance →



* More data will likely help.

* What to do next →

1. More many examples → High variance.
2. Smaller set of fatures → High variance.
3. Additional features → High bias.
4. Polynomial features → High bias
5. Decreasing λ → High bias.
6. Increasing λ → High variance.