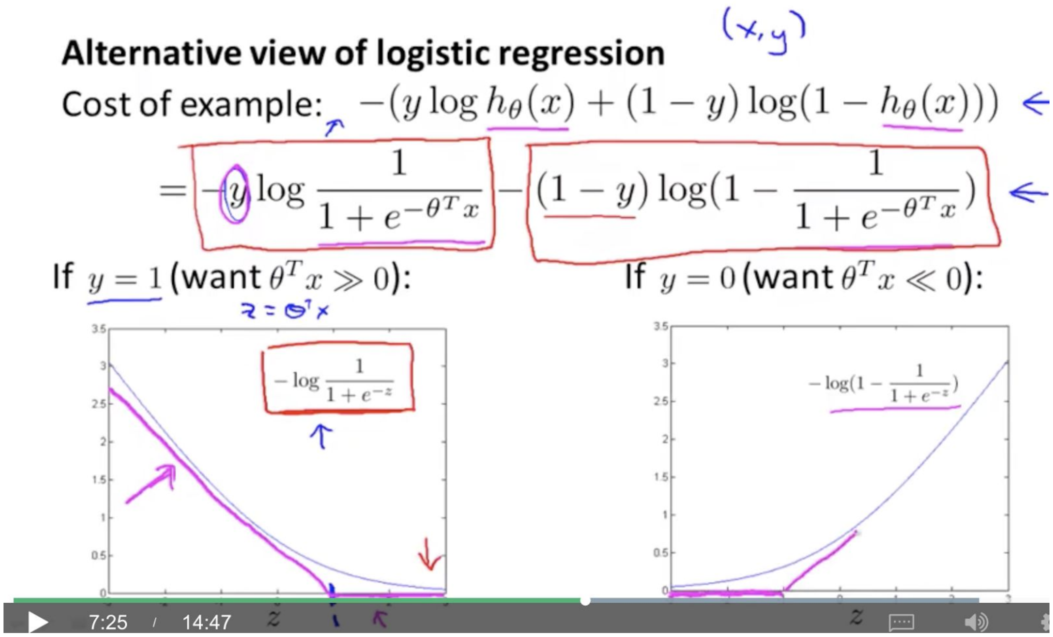
Week 7: Support Vector Machines

Large Margin Classification

Optimization Objective

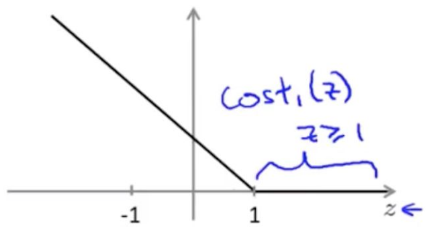


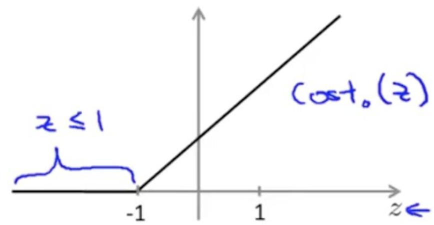
$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

Large Margin Intuition

Support Vector Machine

$$\rightarrow \min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} \underline{cost_1(\theta^T x^{(i)})} + (1 - y^{(i)}) \underline{cost_0(\theta^T x^{(i)})} \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$





→ If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

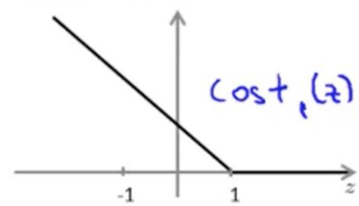
→ If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

SVM Decision Boundary

$$\min_{\theta} C \underbrace{\sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right]}_{=0} + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

Whenever $y^{(i)} = 1$:

$\theta^T x^{(i)} \geq 1$



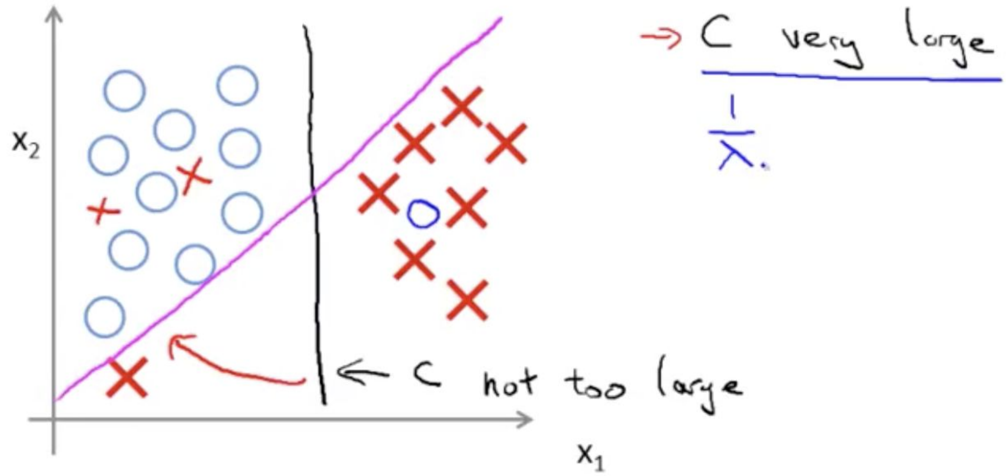
Whenever $y^{(i)} = 0$:

$\theta^T x^{(i)} \leq -1$

$$\min_{\theta} \cancel{C \times 0} + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

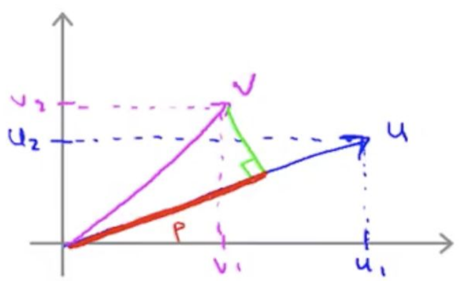
s.t. $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$

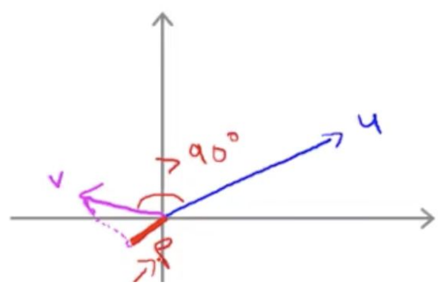
$\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$.



Mathematics Behind Large Margin Classification

Vector Inner Product





$u = \begin{bmatrix}u_1\\u_2\end{bmatrix}$ $\quad v = \begin{bmatrix}v_1\\v_2\end{bmatrix}$

$u^Tv = ?$ $\quad [u_1 \;\; u_2]\begin{bmatrix}v_1\\v_2\end{bmatrix}$

$\|u\|$ = length of vector u

$= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$

p = length of projection of v onto u.

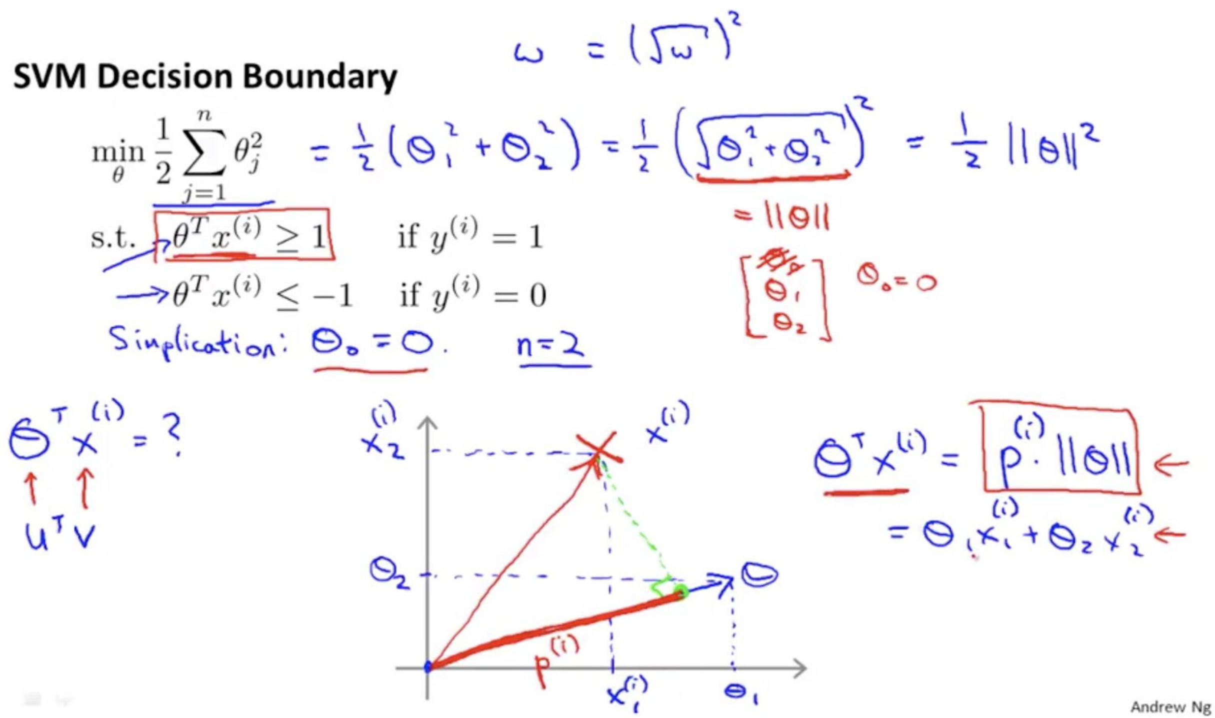
Signed

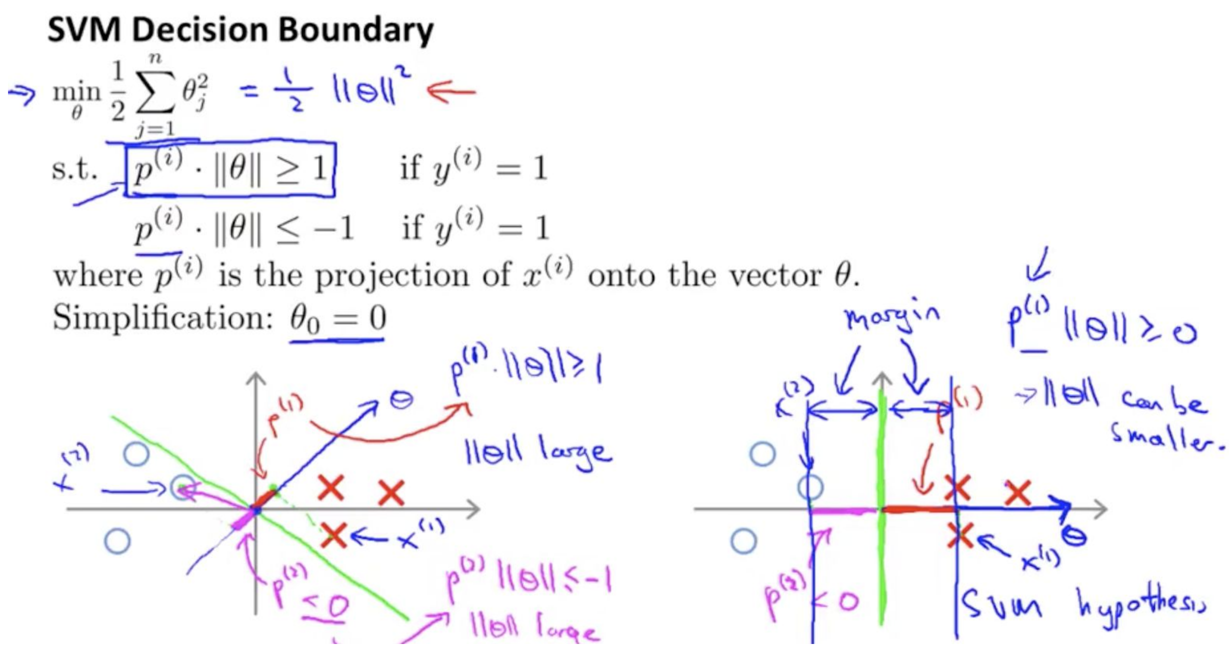
$u^Tv = \underline{p} \cdot \underline{\|u\|}$ $\quad = v^Tu$

$= u_1v_1 + u_2v_2$ $\quad p \in \mathbb{R}$

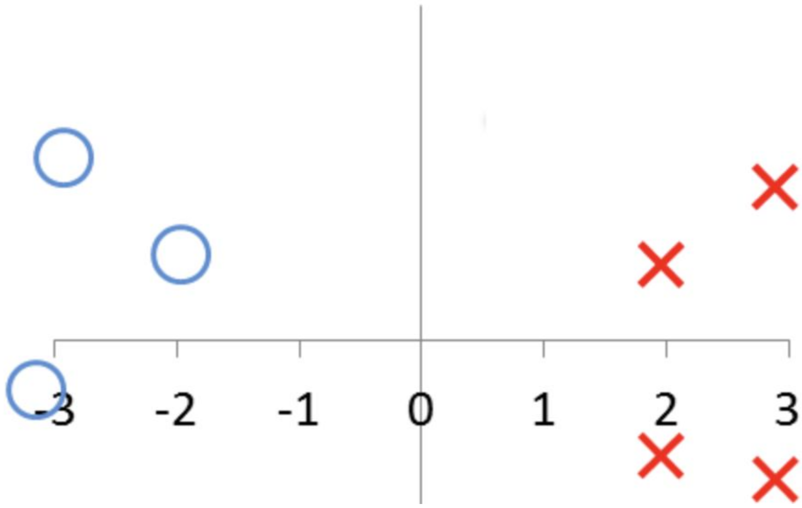
$u^Tv = p \cdot \|u\|$

$p < 0$



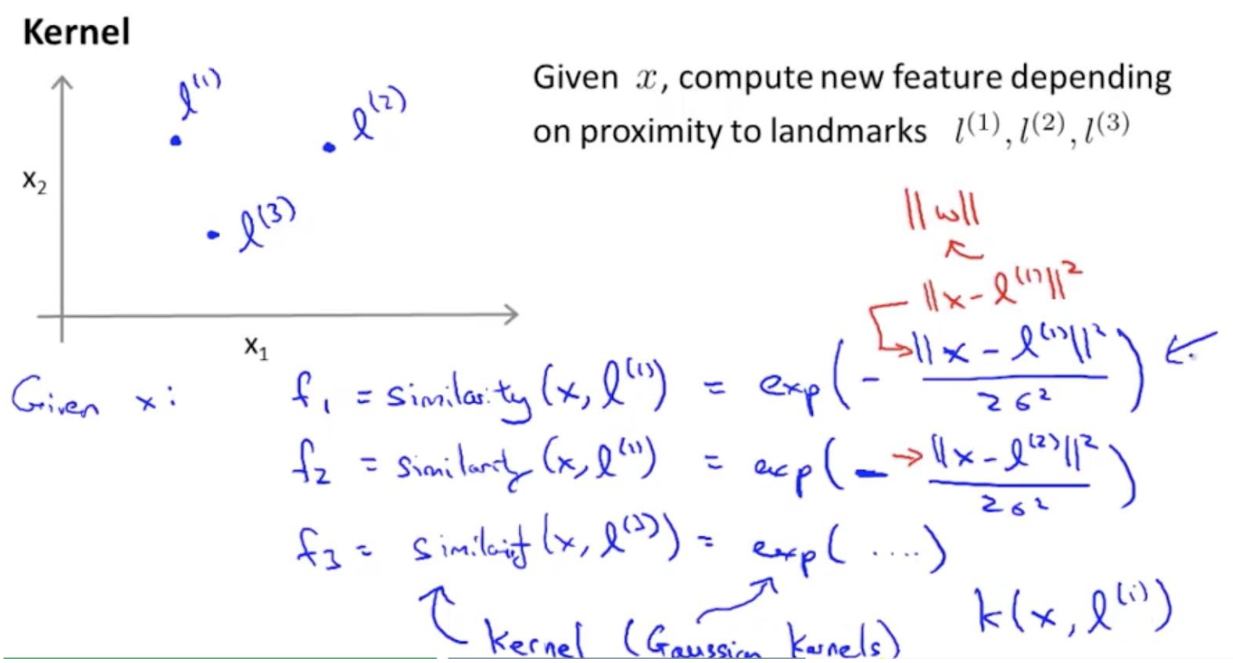


θ_0=0: decision boundary passes the origin.



$$||\theta|| = \frac{1}{2}$$

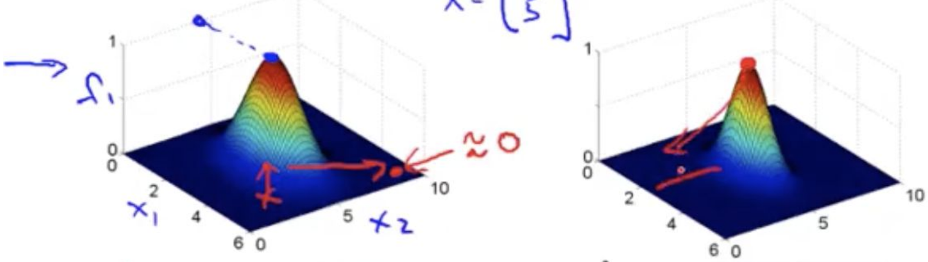
Kernels



Example:

$$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad \underline{f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\boxed{\sigma^2}}\right)}$$

$\sigma^2 = 1$ $\qquad x = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$ $\qquad \sigma^2 = 0.5$



When σ is small, the height does not change while it narrows down.

SVM with Kernels

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,

choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

Given example $\underline{x}$:

$\boxed{f_1} = \text{similarity}(x, l^{(1)})$ $\leftarrow x^{(i)}$

$f_2 = \text{similarity}(x, l^{(2)})$

$\ldots$

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \qquad f_0 = 1$$

For training example $\underline{(x^{(i)}, y^{(i)})}$:

$\underline{x^{(i)}} \rightarrow$

$f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)})$

$f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)})$

$\vdots$

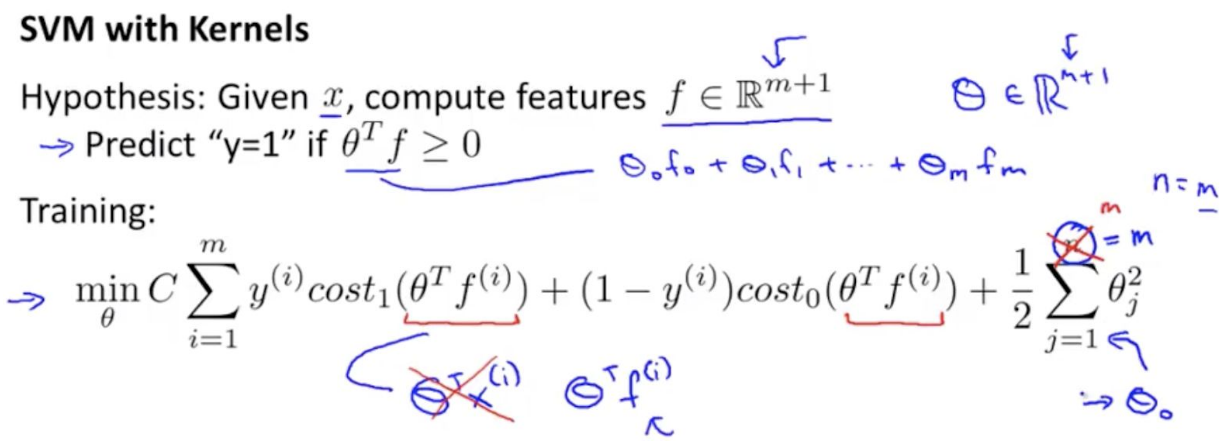
$f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = \exp(-\frac{0}{2\sigma^2}) = 1$ $\quad \swarrow x^{(i)}$

$f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)})$

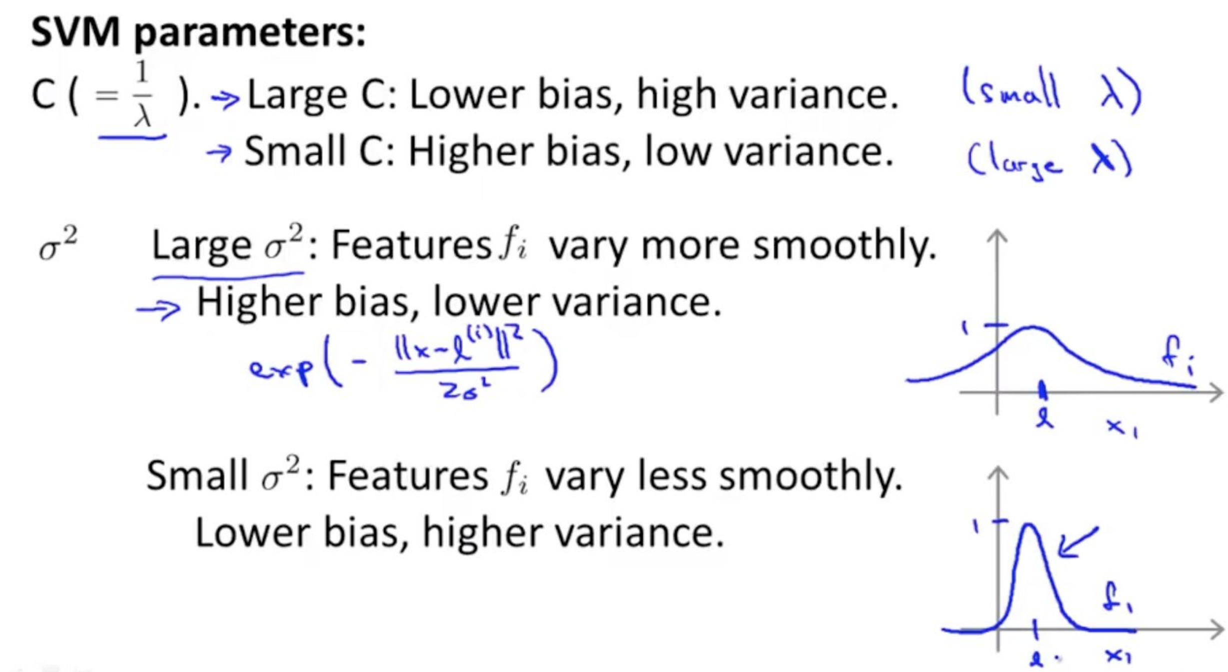
$\underline{x^{(i)} \in \mathbb{R}^{n+1}}$ (or $\mathbb{R}^n$)

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix} \qquad f_0^{(i)} = 1$$

Andrew Ng



Using kernels for logistic regression is very slow, because it runs very slowly.



Suppose you train an SVM and find it overfits your training data. Which of these would be a reasonable next step?

Decrease C and/or Increase sigma square

SVM in Practice

Package: liblinear, libsvm,

Specify:

parameter C;

kernel (similarity function);

E.g. No kernel ("linear kernel")
Predict "y = 1" if $\theta^T x \geq 0$ $\theta_0 + \theta_1 x_1 +$ → n large,

Gaussian kernel:

$$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right), \text{ where } l^{(i)} = x^{(i)}.$$

Need to choose σ^2.

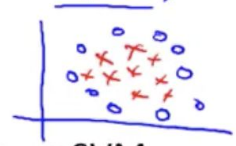
Other kernels: polynomial kernel, string kernel, chi-square kernel, histogram intersection kernel…

Note: not all similarity functions made valid kernel, need to satisfy "mercer's theorem" to make sure SVM packages' optimizations run correction, and do not diverge.

Logistic regression vs. SVMs

n = number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples

- If n is large (relative to m): (E.g. $n \geq m$, $n = 10{,}000$, $m = 10 \cdots 1000$)
- Use logistic regression, or SVM without a kernel ("linear kernel")
- If n is small, m is intermediate: ($n = 1 - 1000$, $m = 10 - 10{,}000$)
  → Use SVM with Gaussian kernel



If n is small, m is large: ($n = 1 - 1000$, $m = 50{,}000+$)
→ Create/add more features, then use logistic regression or SVM without a kernel

Neural network likely to work well for most of these settings, but may be slower to train; SVM does not need to worry about local minimum.

Quiz:

Suppose you have 2D input examples (ie, $x^{(i)} \in \mathbb{R}^2$). The decision boundary of the SVM (with the linear kernel) is a straight line.

Week 8:

Clustering

Application: market segmentation; Social network analysis; organize computing clusters; astronomical data analysis

K-Means Algorithm

K-means algorithm

Input:

- K (number of clusters)
- Training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

K-means algorithm

μ_1 × μ_2 ×

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$

Repeat {

Cluster assignment step

for i = 1 to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

$\min_k ||x^{(i)} - \mu_k||^2 \rightarrow c^{(i)}$

Move centroid

for k = 1 to K

μ_k := average (mean) of points assigned to cluster k

$x^{(1)}, x^{(5)}, x^{(6)}, x^{(10)} \rightarrow c^{(1)}=2, c^{(5)}=2, c^{(6)}=2, c^{(10)}=2$

$$\mu_2 = \frac{1}{4}\left[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}\right] \in \mathbb{R}^n$$

}

K: total number of clusters; k: the index of cluster

Optimization Objective

Usage:

debug the learning algorithm to ensure the K-Means is running correctly;

Find better costs for this and avoid the local ultima

Random Initialization

- Should have K < m
- Randomly pick K training examples
- Set μ_1, … μ_K equal to these K examples.μ_1, … μ_K equal to these K examples.

Random initialization

For i = 1 to 100 {   50 – 1000

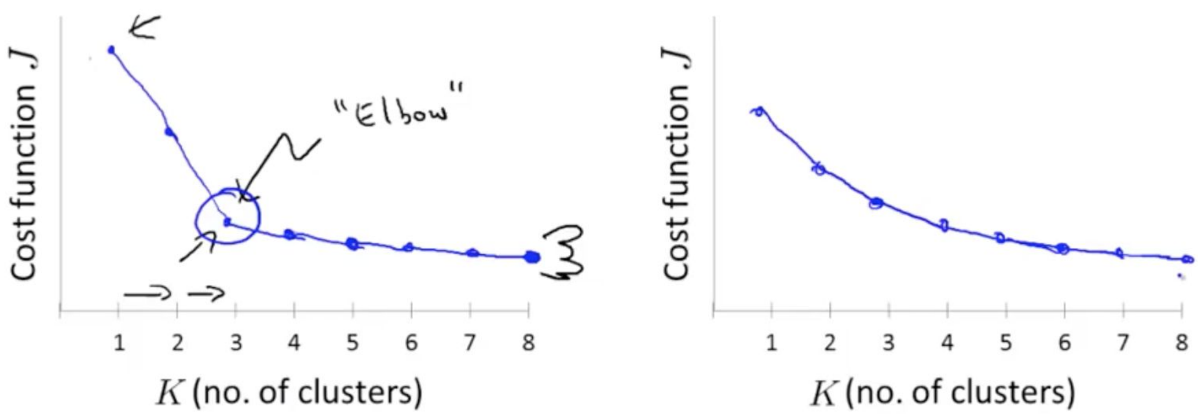
Randomly initialize K-means.
Run K-means. Get $c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K$.
Compute cost function (distortion)
$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$
}

Pick clustering that gave lowest cost $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$

Choosing the number of Clusters

Choosing the value of K

Elbow method:



Dimensionality Reduction Motivation

Data Compression

- Reduce memory/disk needed to store data
- Speed up learning algorithm

Visualization

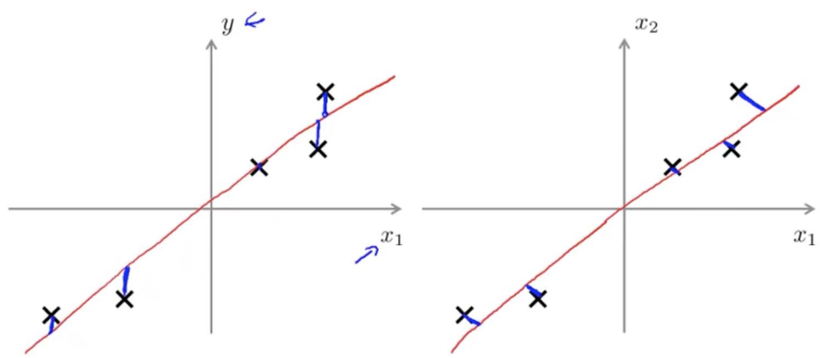
Principal Component Analysis

PCA Problem Formulation

Minimize the projection error

The left is linear regression; and the right is PCA

PCA is not linear regression



Data preprocessing

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

Preprocessing (feature scaling/mean normalization):

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

Replace each $x_j^{(i)}$ with $x_j - \mu_j$.

If different features on different scales (e.g., $x_1 =$ size of house, $x_2 =$ number of bedrooms), scale features to have comparable range of values.

PCA Algorithm

Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n} \underbrace{\boxed{\underbrace{(x^{(i)})}_{n\times 1}\underbrace{(x^{(i)})^T}_{1\times n}}}_{n\times n}$$

Sigma

Compute "eigenvectors" of matrix Σ:

```
[U,S,V] = svd(Sigma);
```

n×n matrix.

→ Singular value decomposition

eig(Sigma)

Since Σ is symmetrical, we can get the same answer from both svd and eig.

Principal Component Analysis (PCA) algorithm

From `[U,S,V] = svd(Sigma)` , we get:

$$\rightarrow U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n\times n}$$

k

$$x \in \mathbb{R}^n \longrightarrow z \in \mathbb{R}^k$$

$$z = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & & | \end{bmatrix}^T}_{n\times k} x = \underbrace{\underbrace{\begin{bmatrix} — (u^{(1)})^T — \\ \vdots \\ — (u^{(k)})^T — \end{bmatrix}}_{k\times n} \underbrace{x}_{n\times 1}}_{k \times 1}$$

$z \in \mathbb{R}^k$

U_{reduce}

Principal Component Analysis (PCA) algorithm summary

→ After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

$$\texttt{Sigma} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$$

$$X = \begin{bmatrix} — x^{(1)T} — \\ \vdots \\ — x^{(m)T} — \end{bmatrix}$$

$$\boxed{\texttt{Sigma = (1/m) * X' * X;}}$$

```
→ [U,S,V] = svd(Sigma);
→ Ureduce = U(:,1:k);
→ z = Ureduce'*x;
```

$x \in \mathbb{R}^n$ $\quad \cancel{x_0 = 1}$

Applying PCA

Reconstruction from Compressed Representation

$$z \in \mathbb{R} \rightarrow x \in \mathbb{R}^2$$

$$\tilde{x}^{(i)} \;\; x_{approx}^{(i)} = \underbrace{\underbrace{U_{reduce}}_{n\times k} \cdot \underbrace{z^{(i)}}_{k\times 1}}_{n\times 1}$$

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

Choosing the Number of Principal Components

Choosing k (number of principal components)

Average squared projection error: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2$

Typically, choose k to be smallest value so that

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01 \qquad \text{(1\%)}$$

"99% of variance is retained"

Choosing k (number of principal components)

Algorithm:

Try PCA with $\cancel{k=1}$ $\cancel{k=2}$ $\cancel{k=3}$ $k=4$ ...

Compute $U_{reduce}, z^{(1)}, z^{(2)}, \ldots, z^{(m)}, x_{approx}^{(1)}, \ldots, x_{approx}^{(m)}$

Check if

$$\boxed{\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2}} \leq 0.01?$$

$k = 17$

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

$$S = \begin{bmatrix} S_{11} & & & 0 \\ & S_{22} & & \\ & & S_{33} & \\ 0 & & & \ddots \; S_{nn} \end{bmatrix}$$

For given k

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

$k=3$

Advice for Applying PCA

Bad use of PCA: To prevent overfitting

→ Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $\overset{1000}{k} < n$. — 10000

Thus, fewer features, less likely to overfit.

Bad!

This might work OK, but isn't a good way to address overfitting. Use regularization instead.

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \boxed{\frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2}$$

PCA is sometimes used where it shouldn't be

Design of ML system:

- → Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
- → Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$
- → Train logistic regression on $\{(z^{(1)}, y^{(1)}), \ldots, (z^{(m)}, y^{(m)})\}$
- → Test on test set: Map $x_{test}^{(i)}$ to $z_{test}^{(i)}$. Run $h_\theta(z)$ on $\{(z_{test}^{(1)}, y_{test}^{(1)}), \ldots, (z_{test}^{(m)}, y_{test}^{(m)})\}$

→ How about doing the whole thing without using PCA?

→ Before implementing PCA, first try running whatever you want to do with the original/raw data $x^{(i)}$. Only if that doesn't do what you want, then implement PCA and consider using $z^{(i)}$.

Quiz:

If the input features are on very different scales, it is a good idea to perform feature scaling before PCA