
Machine Learning : Andrew NG

Machine learning

- Arthur Samuel (1958)
- Tom Mitchell : E/T/P

Given data,

Supervised

→ "right answers" given

→ Types of supervised learning problems

- Regression (continuous-valued outputs)
- Classification (discrete)

Unsupervised

→ little/no info abt "right answers"

→ derive structure from data, where we don't know effect of variables

→ no feedback based on prediction results

- Clustering
- Non-clustering
  - Cocktail party problem algorithm
  - Structure in chaotic env.

Supervised learning problem: given a training set, to learn a function $h: x \to y$, so that $h(x)$ is a "good" predictor for corresponding value of y

Training set: m, x, y

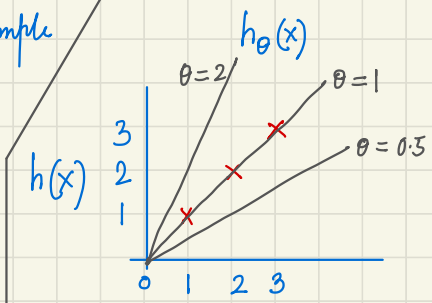
(x, y) : one training example

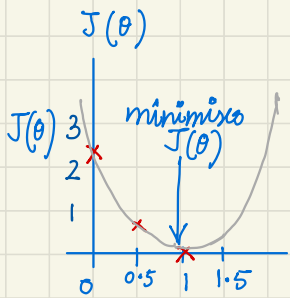
(x^i, y^i) : i^{th} training example

Learning algorithm

$x \to h \to$ predicted y

(h: hypothesis)





Problem: Univariate Linear Regression

Hypothesis : $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters : θ_0, θ_1

→ for fixed θ, f^n of x → func^n of θ

Cost function : $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta(x^i) - y^i \right]^2$

→ Minimisation algorithm : Gradient Descent

Goal : minimise $J(\theta_0, \theta_1)$ over θ_0, θ_1

→ Repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

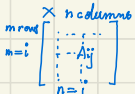
}

simultaneous update

- α : learning rate
  - adjust : failure / long time to converge
- $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$: derivatives

Assignment	vs	Truth assertion
$a := b$		$a = b$
$a := a + 1$		$a = a + 1$ ✗

Linear Algebra Review:



Matrices and Vectors

- **Matrix**: 2-dimensional array of numbers, where elements $A_{ij} \in \mathbb{R}^{m \times n}$
- **Vector**: $n \times 1$ matrix $\equiv$ n-dim vector $\quad y_i \in \mathbb{R}^n$
- **Scalar**: object is a single value

- Matrix addition and subtraction
  - same dimension / element-wise / BODMAS
- Matrix Multiplication
  - Matrix-vector multiplication
  - matrix-matrix multiplication
  - Properties:
    - $AB \neq BA$ (non-commutative)
    - $(AB)C \neq A(BC)$ (non-associative)
    - $A \cdot I = I \cdot A = A$ (Identity matrix) dimension implicit

$$\begin{matrix} m \times n & n \times n & m \times m & m \times n \end{matrix}$$

$$\begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix}$$

- Inverse and Transpose
  - Matrix Inverse: If $A = m \times m$ and has inverse (singular matrix ✗)
    $$AA^{-1} = A^{-1}A = I$$
  - Transpose: $A \rightarrow A^T$
    $$A_{ij} = A^T_{ji}$$

One Hypothesis:

House Sizes	Hypothesis
→ 2104	$h_\theta(x) = -40 + 0.25x$
→ 1416	
→ 1534	
→ 852	

$$\underbrace{\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix}}_{4 \times 2} \times \underbrace{\begin{bmatrix} -40 \\ 0.25 \end{bmatrix}}_{2 \times 1} = \underbrace{\begin{bmatrix} \cdots \\ \cdots \\ \cdots \\ \cdots \end{bmatrix}}_{4 \times 1}$$

Matrix Vector $h_\theta(x)$

Data matrix × parameters = Prediction

Competing hypothesis:

House sizes:

Hypothesis:
1. $h_\theta(x) = \ldots$
2. $h_\theta(x) = \ldots$
3. $h_\theta(x) = \ldots$

Matrix × Matrix = Prediction

Multi-variate linear regression

Multiple features: n, $x^{(i)}_j$ (i^{th} training example, j^{th} feature)

n: # features

m: # training examples

Univariate → multivariate

x: scalar $\quad x$: vector

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

$$= \sum_{i=1} \theta_i x_i \quad \text{Define } x_0 = 1 \quad = \theta^T x$$

→ Gradient descent for multiple variables:

repeat until convergence {
$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_j^{(i)}$ for $j = 0, \ldots n$
}

$x_i \to \frac{x_i - \mu_i}{s_i}$

+ FEATURE SCALING $(-1 \le x_i \le 1)$
+ MEAN NORMALISATION: $x_i \to x_i - \mu_i$

μ_i: training set mean
s_i: " std dev/range

+ DEBUGGING: $J(\theta)$ vs # iterations → slow convergence, α too small
  + Automatic convergence test
  → $J(\theta)$ ↑, non-convergence, α too large
+ LEARNING RATE Try 0·001, 0·003, 0·01, 0·03, 0·1, ...

→ Linear Regression vs Polynomial Regression

$h_\theta(x) = \theta_0 + \theta_1 x$

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

$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

$= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

converting polynomial → linear problem

Making new features by combining

$x_1 = x$
$x_2 = x^2$
$x_3 = x^3$
Ranges change

Choice depends on specific training set/example

Minimisation algorithms: for $J(\theta)$

→ Gradient Descent
→ Normal Equation:

$\theta \in \mathbb{R}^{n+1}$, $J(\theta_0, \theta_1, \ldots, \theta_n)$

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

$= \frac{1}{2m} (X\theta - y)^T (X\theta - y)$

$\frac{\partial}{\partial \theta_j} J(\theta) = \ldots\ldots \overset{\text{set}}{=} 0 \quad \forall j$

Solve for $\theta_0, \theta_1, \ldots \theta_n$

$$\theta = (X^T X)^{-1} X^T y$$

per training eg $x^{(i)} = \begin{bmatrix} x_0^i \\ x_1^i \\ \vdots \\ x_n^i \end{bmatrix} \in \mathbb{R}^{n+1}$

X: Design matrix $= \begin{bmatrix} \cdots (x^{(1)})^T \cdots \\ \cdots (x^{(2)})^T \cdots \\ \vdots \\ \cdots (x^{(m)})^T \cdots \end{bmatrix}$ $m \times (n+1)$

Gradient Descent	Normal Eqn
→ iterative minimisation of $J(\theta)$	→ explicitly & analytically compute $\frac{\partial J(\theta)}{\partial \theta_j} = 0$ to solve for $\theta_{j,\min}$ Use Normal Eqn $\theta = (X^T X)^{-1} X^T y$
→ choose α	→ no need for α
→ needs many iters	→ no need for iters
→ $O(kn^2)$	→ $O(n^3)$, need to compute $(X^T X)^{-1}$
→ works well for large N	→ computationally limited

ML
- Supervised
  - Regression
  - Classification / Logistic Regression
- Unsupervised

CLASSIFICATION / Logistic Regression

- **Binary classification:** $y \in \{0,1\} \equiv \{-,+\}$ → Want: $0 \le h_\theta(x) \le 1$
  - y: training example label

Hypothesis Representation

Model: $h_\theta(x) = g(\theta^T x)$, $g(z) = \frac{1}{1+e^{-z}}$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

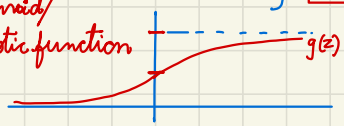
Interpretation → estimated probability that $y=1$ on input x, parameterised by θ

$$h_\theta(x) = p(y=1 \mid x;\theta)$$

→ $p(y=0|x,\theta) + p(y=1|x,\theta) = 1$

$p(y=0|x,\theta) = 1 - p(y=1|x,\theta)$

Sigmoid / Logistic function



$h_\theta(x) \ge 0.5 \rightarrow y=1 \leftarrow \theta^T x \ge 0$

$h_\theta(x) < 0.5 \rightarrow y=0 \leftarrow \theta^T x < 0$

Decision boundary: $h_\theta(x) = 0.5 \iff \theta^T x = 0$

- property of hypothesis function
- linear / polynomial

Cost function

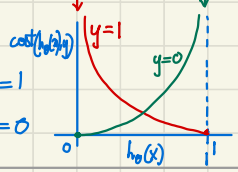
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{cost}(h_\theta(x), y)$$

where, $\text{cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log(1-h_\theta(x)) & \text{if } y=0 \end{cases}$

penalise when prediction with abs. certainty turns out to be wrong!

for $y=1$, $\text{cost}(h_\theta(x), y) \to \infty$ as $h_\theta(x) \to 0$

for $y=0$, $\text{cost}(h_\theta(x), y) \to \infty$ as $h_\theta(x) \to 1$



$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right]$$

$$= \frac{1}{m}\left(-y^T \log(h) - (1-y)^T \log(1-h)\right)$$

→ Minimise $J(\theta)$ using GRADIENT DESCENT

Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$$ [vectorised implementation]

}

looks identical to linear regression but

$h_\theta(x) = \theta^T x$ vs $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

[Can also use ready-to-use advanced algorithms]

- Multi-class classification: One vs All (one-vs-rest)

$y \in \{0, 1, \ldots, n\}$

→ Train logistic regression classifier / hypothesis $h_\theta^{(i)}(x) = p(y = i \mid x, \theta)$ for each class i

→ For new input x, to make prediction, pick class i which maximises: $\max_i h_\theta^{(i)}(x)$

Underfitting / high bias vs **Overfitting / high variance**

- hypothesis doesn't map well to data trend
- too simple
- too few features

- fits data but doesn't generalise to predict new data
- too complicated
- too many

Solutions:

1. ↓ # features
   - → manual
   - → model selection
2. Regularisation
   - → smaller values for params: $\theta_0 \ldots \theta_n$
   - "simpler" hypo
   - Less prone to overfitting

→ Takes care of non-invertibility

REGULARISATION

→ LINEAR REGRESSION

- Regularised cost function

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 + \lambda \sum_{j=1}^{m} \theta_j^2\right]$$

λ: Regularisation parameter

- Algorithms

→ Gradient Descent:

Repeat {

$\theta_0 := \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$

$\theta_j := \left(1 - \frac{\alpha}{m}\lambda\right)\theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_j^i \quad \forall j \neq 0$

}

→ Normal Equation

$\theta = (X^T X + \lambda \cdot L)^{-1} X^T y$

where $L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix}$ $(n+1) \times (n+1)$

Non-invertibility:

if $m < n$

$X^T X$ non-inv

but $X^T X + \lambda \cdot L$ is inv.

→ LOGISTIC REGRESSION

- Regularised Cost function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{n}\left[y^i \log(h_\theta(x^i)) + (1 - y^i)\log(1 - h_\theta(x^i))\right] + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

- Algorithms:
  - → Gradient descent: Same as linear regression, but w/ different hypothesis
  - → Advanced optimisation algorithms

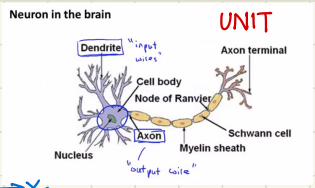
NEURAL NETWORKS

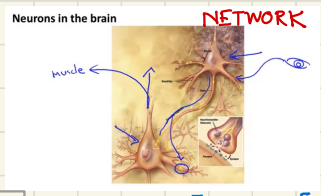
Learning algorithms:

- Linear Regression [Regression, eg, housing prices]
- Logistic Regression [Classification, eg, tumor prediction]
- Neural Networks [Non-linear hypothesis]

} not scalable to non-linear hypothesis with many features

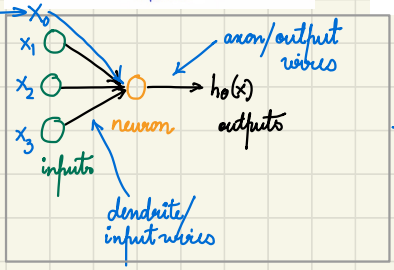
↑ Approximation of brain's learning algorithm





"bias unit" $= 1$

Single Neuron Neural Model: Logistic Model



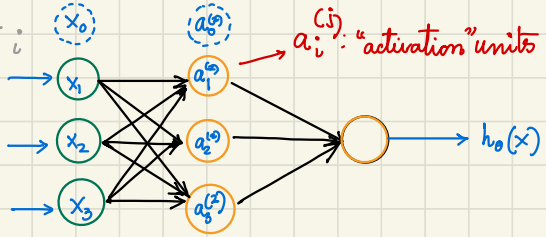
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$\Rightarrow h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

Sigmoid (logistic) activation function

"weights" / params

Neural Network Model



Simplistic Representation

$[x_1 x_2 \dots x_n] \rightarrow [\ \ ] \rightarrow h_\theta(x)$

$a^{(1)} = x$ (add $a_0^{(1)}$)

$z^{(2)} = \Theta^{(1)} a^{(1)}$

$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

$z^{(3)} = \Theta^{(2)} a^{(2)}$

$a^{(3)} = g(z^{(3)}) = h_\theta(x)$

Layer j: Layer 1 / Input layer → Layer 2 / Hidden layers → Layer 3 / output layer

Forward propagation: Vectors implementation

$\Theta^{(j)}$: matrix of weights controlling function mapping from layer $j \rightarrow j+1$

Network: layer $j \rightarrow s_j$ units

$j+1 \rightarrow s_{j+1}$ units

$\Theta^{(j)} \in \mathbb{R}^{s_{j+1} \times (s_j + 1)}$

Generalising:

$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$, $z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ \vdots \\ z_n^{(j)} \end{bmatrix}$ → Setting $x = a^{(1)}$ → $z^{(j)} = \Theta^{(j-1)} a^{(j-1)}$

$z^{(j)}$: $s_j \times 1$; $\Theta^{(j-1)}$: $s_j \times (n+1)$; $a^{(j-1)}$: $(n+1) \times 1$

Examples and Intuitions:

→ Construct fundamental operations in computers (AND/OR/XOR/XNOR) using neural networks

Layer $j \Rightarrow a^{(j)}_{s_j \times 1} = g(z^{(j)}_{s_j \times 1})$ [element-wise]

Add bias unit to layer j: $a_0^{(j)}$

$a^{(j)} \in \mathbb{R}^{s_j} \rightarrow \mathbb{R}^{s_j+1}$

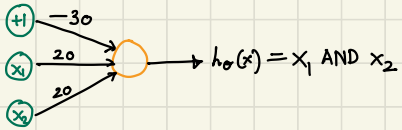
Layer: $j+1 \Rightarrow z^{(j+1)} = \Theta^{(j)} a^{(j)}$

$z^{(j+1)}$: 1×1; $\Theta^{(j)}$: $1 \times (s_j+1)$; $a^{(j)}$: $(s_j+1) \times 1$

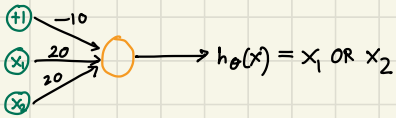
Final step: $h_\theta(x) = a^{(j+1)} = g(z^{(j+1)})$

Same as logistic Regression

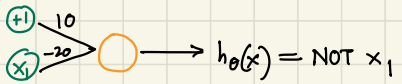
AND

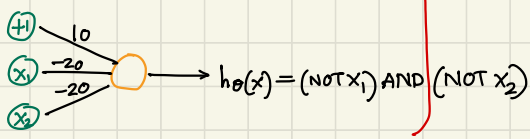


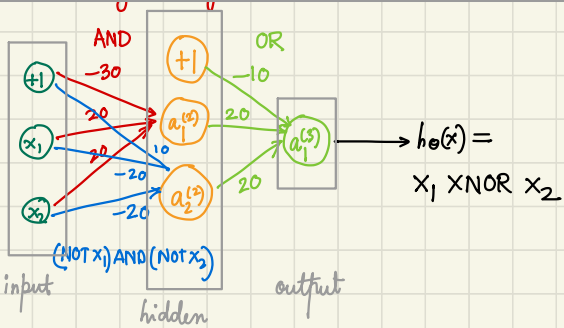
OR



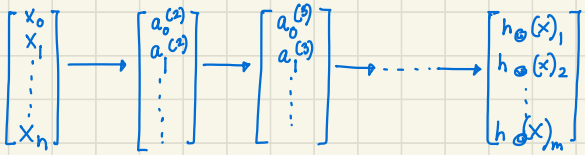
NEGATION







Multi-class Classification



COST FUNCTION [Neural Networks]

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)} \log h_\Theta(x^{(i)})_k + (1-y_k^{(i)}) \log (1 - h_\Theta(x^{(i)})_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}} (\Theta_{j,i}^{(l)})^2$$

L: total # layers in the network
S_l: # units (not counting bias unit) in layer l
K: # output units/classes
$h_\Theta(x)_k$: hypothesis resulting in k^{th} output

(i,j,k,l): unit in l^{th} layer, unit in $(l+1)^{th}$ layer, output, layer.
↳ $\Theta_{j,i}^{(l)} \in \mathbb{R}$

Gradient Computation: Backpropagation

≡ Neural network terminology for minimising cost function

BP in NNs ≡ Gradient descents in linear/logistic regression

Algorithms:
→ gradient descent
↳ advanced (fminunc, fmincg)
↳ BP
→ normal eq^n's

Goal: $\min_\Theta J(\Theta)$ → Compute $\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta)$

→ Backward propagation algorithm

→ Given training set $\{(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})\}$

- Set $\Delta_{i,j}^{(l)} := 0 \quad \forall\, i,j,l$ (initialising)

→ For training example $t = 1 \to m$,

1. Set $a^{(1)} := x^{(t)}$
2. Perform FP to compute $a^{(l)}$ for $l = 2,3,\dots L$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)}) \quad (\text{add } a_0^{(l+1)})$$

3. Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$ (error values)
4. Perform BP to compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots \delta^{(2)}$, where:

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$$

$$\| \quad g'(z^{(l)})$$

5. Update Δ matrix:

$$\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)} \quad \text{or,} \quad \Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

$$\Rightarrow \begin{cases} D_{i,j}^{(l)} := \frac{1}{m}\left(\Delta_{i,j}^{(l)} + \lambda \Theta_{j,i}^{(l)}\right) & j \neq 0 \\ D_{i,j}^{(l)} := \frac{1}{m}\Delta_{i,j}^{(l)} & j = 0 \end{cases}$$

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J_\Theta = D_{i,j}^{(l)}$$

Learning Algorithm
→ Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.
→ Unroll to get `initialTheta` to pass to
→ `fminunc(@costFunction, initialTheta, options)`

```
function [jval, gradientVec] = costFunction(thetaVec)
    From thetaVec, get Θ(1), Θ(2), Θ(3).
    Use forward prop/back prop to compute D(1), D(2), D(3) and J(Θ)
    Unroll D(1), D(2), D(3) to get gradientVec.
```

[2] One effective strategy for choosing ϵ_{init} is to base it on the number of units in the network. A good choice of ϵ_{init} is $\frac{\sqrt{6}}{\sqrt{L_{in}+L_{out}}}$, where $L_{in} = s_l$ and $L_{out} = s_{l+1}$ are the number of units in the layers adjacent to Θ_l.

MACHINE-LEARNING DIAGNOSTICS

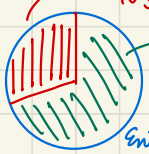
Basic trouble-shooting techniques:

- more training examples → fixes high variance
- trying smaller set of features (counter overfitting) → fixes high variance
- trying additional features (counter underfitting) → fixes high bias
- trying polynomial features (include non-linearities) → fixes high bias
- ↑ or ↓ λ → fixes high bias/variance

→ **Evaluating Hypothesis:**

→ learn Θ and min $J_{train}(\Theta)$ using training set

→ compute test set error $J_{test}(\Theta)$

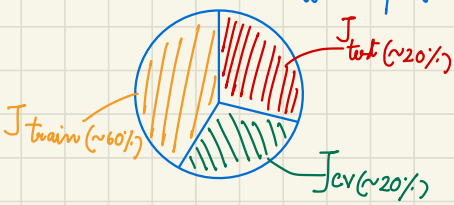


$$err(h_\Theta(x), y) = \begin{cases} 1 & \text{if } h_\Theta(x) \geq 0.5 \ \& \ y = 0 \text{ or } h_\Theta(x) < 0.5 \ \& \ y = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$J_{test}(\Theta) = \begin{cases} \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left(h_\Theta(x_{test}^{(i)}) - y_{test}^{(i)} \right)^2 & \text{[linear regression]} \\ \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err\left(h_\Theta(x_{test}^{(i)}), y_{test}^{(i)} \right) & \text{[classification]} \end{cases}$$

(proportion of cases misclassified)

→ **Model Selection:** different polynomial degrees (d)



→ optimise Θ for different d for training set

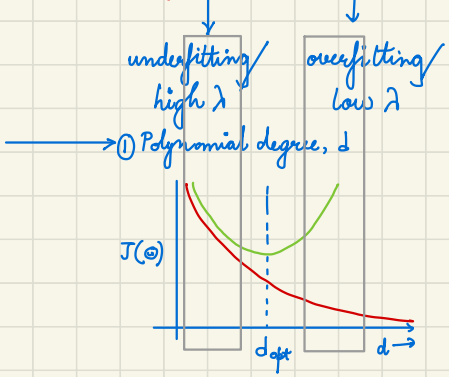
→ use CV to find d with least J_{CV}

→ estimate generalisation error using $J_{test}(\Theta^{(d)})$

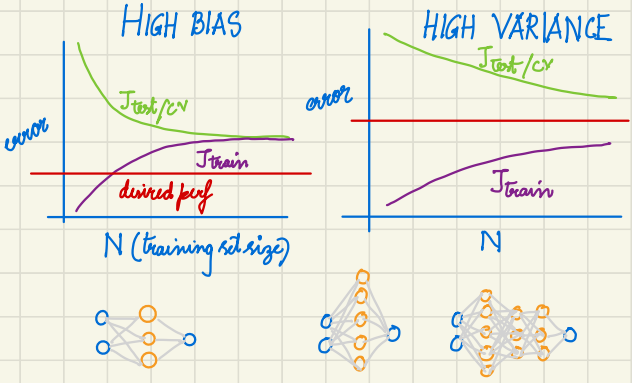
↓

polynomial is not trained using test set

→ **Diagnosing Bias vs Variance**



LEARNING CURVES



computationally cheap but prone to underfitting

expensive but prone to overfitting → Use regularisation to counter

→ ② Regularisation
- create λ list, $\lambda \in 0, 0.1, 0.2, 0.4 \ldots 10.24$
- list of models with different d
- Iterate through λ, and $\forall \lambda$, through d, $J_{train}(\Theta)$
- Compute $J_{cv}(\Theta_{train})$ with $\lambda = 0$ (use λ)
- Select (λ, d) combo for least J_{cv}
- generalise to J_{test}

BUILDING A SPAM CLASSIFIER

→ Prioritising what to work on

→ Recommended approach

└→ quick and dirty implementation on CV set + learning curves (pre-optimisation)

└→ Error analysis (manually examine)

└→ common types of misclassified examples (helps what to focus on first), eg pharma (12), replica (4), steal password (53)

└→ cues what might help prevent these misclassifications, eg, misspellings (5), unusual punctuation (32)

└→ Numerical evaluation, eg, "Porter stemmer"/stemming

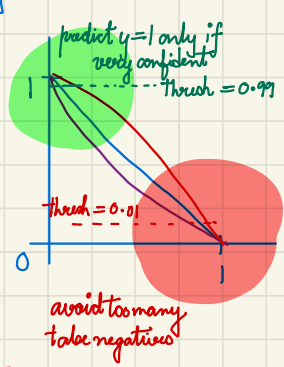
SKEWED DATA

$y=1$: rare case

Predicted \ Actual	1	0
1	True Pos	False Pos
0	False Neg	True Neg

→ Precision $= \frac{\text{True Pos}}{\#\text{Predicted pos}} = \frac{\text{True Pos}}{\text{True Pos} + \text{False pos}}$

→ Recall $= \frac{\text{True Pos}}{\#\text{actual pos}} = \frac{\text{True Pos}}{\text{True pos} + \text{False Neg}}$



Predict $y=1$ if $h_\Theta(x) \geq$ threshold

F_1 Score : $\frac{2PR}{P+R}$

F-Score on CV set $= \begin{cases} 0 & \text{if } P=0 \text{ or } R=0 \\ 1 & \text{if } P=1 \text{ \& } R=1 \end{cases}$

LARGE DATA SETS

→ Rationale: for low bias algorithm, larger dataset = lesser chances of overfitting

└→ "enough" info: tip: given input x, will human expert be able to confidently predict y ?

Optimization Objective

The **Support Vector Machine** (SVM) is yet another type of *supervised* machine learning algorithm. It is sometimes cleaner and more powerful.

Recall that in logistic regression, we use the following rules:

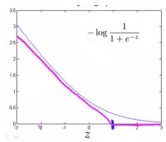
if y=1, then $h_\theta(x) \approx 1$ and $\Theta^T x \gg 0$

if y=0, then $h_\theta(x) \approx 0$ and $\Theta^T x \ll 0$

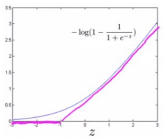
Recall the cost function for (unregularized) logistic regression:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))$$
$$= \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}\log\left(\frac{1}{1+e^{-\theta^T x^{(i)}}}\right) - (1-y^{(i)})\log\left(1-\frac{1}{1+e^{-\theta^T x^{(i)}}}\right)$$

To make a support vector machine, we will modify the first term of the cost function $-\log(h_\theta(x)) = -\log\left(\frac{1}{1+e^{-\theta^T x}}\right)$ so that when $\theta^T x$ (from now on, we shall refer to this as z) is **greater than** 1, it outputs 0. Furthermore, for values of z less than 1, we shall use a straight decreasing line instead of the sigmoid curve.(In the literature, this is called a hinge loss (https://en.wikipedia.org/wiki/Hinge_loss) function.)



Similarly, we modify the second term of the cost function $-\log(1-h_{\theta(z)}) = -\log\left(1-\frac{1}{1+e^{-\theta^T x}}\right)$ so that when z is **less than** -1, it outputs 0. We also modify it so that for values of z greater than -1, we use a straight increasing line instead of the sigmoid curve.



We shall denote these as $\text{cost}_1(z)$ and $\text{cost}_0(z)$ (respectively, note that $\text{cost}_1(z)$ is the cost for classifying when y=1, and $\text{cost}_0(z)$ is the cost for classifying when y=0), and we may define them as follows (where k is an arbitrary constant defining the magnitude of the slope of the line):

$z = \theta^T x$

$\text{cost}_0(z) = \max(0, k(1+z))$

$\text{cost}_1(z) = \max(0, k(1-z))$

Recall the full cost function from (regularized) logistic regression:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{m} y^{(i)}(-\log(h_\theta(x^{(i)}))) + (1-y^{(i)})(-\log(1-h_\theta(x^{(i)}))) + \frac{\lambda}{2m}\sum_{j=1}^{n}\Theta_j^2$$

Note that the negative sign has been distributed into the sum in the above equation.

We may transform this into the cost function for support vector machines by substituting $\text{cost}_0(z)$ and $\text{cost}_1(z)$:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{m} y^{(i)}\,\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\,\text{cost}_0(\theta^T x^{(i)}) + \frac{\lambda}{2m}\sum_{j=1}^{n}\Theta_j^2$$

We can optimize this a bit by multiplying this by m (thus removing the m factor in the denominators). Note that this does not affect our optimization, since we're simply multiplying our cost function by a positive constant (for example, minimizing $(u-5)^2+1$ gives us 5; multiplying it by 10 to make it $10(u-5)^2+10$ still gives us 5 when minimized).

$$J(\theta) = \sum_{i=1}^{m} y^{(i)}\,\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\,\text{cost}_0(\theta^T x^{(i)}) + \frac{\lambda}{2}\sum_{j=1}^{n}\Theta_j^2$$

Furthermore, convention dictates that we regularize using a factor C, instead of λ, like so:

$$J(\theta) = C\sum_{i=1}^{m} y^{(i)}\,\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\,\text{cost}_0(\theta^T x^{(i)}) + \frac{1}{2}\sum_{j=1}^{n}\Theta_j^2$$

This is equivalent to multiplying the equation by $C = \frac{1}{\lambda}$, and thus results in the same values when optimized. Now, when we wish to regularize more (that is, reduce overfitting), we *decrease* C, and when we wish to regularize less (that is, reduce underfitting), we *increase* C.

Finally, note that the hypothesis of the Support Vector Machine is *not* interpreted as the probability of y being 1 or 0 (as it is for the hypothesis of logistic regression). Instead, it outputs either 1 or 0. (In technical terms, it is a discriminant function.)

$$h_\theta(x) = \begin{cases} 1 & \text{if } \Theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Large Margin Intuition

A useful way to think about Support Vector Machines is to think of them as *Large Margin Classifiers*.

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)

Now when we set our constant C to a very **large** value (e.g. 100,000), our optimizing function will constrain Θ such that the equation A (the summation of the cost of each example) equals 0. We impose the following constraints on Θ:

$\Theta^T x \geq 1$ if y=1 and $\Theta^T x \leq -1$ if y=0.

If C is very large, we must choose Θ parameters such that:

$\sum_{i=1}^{m} y^{(i)}\text{cost}_1(\Theta^T x) + (1-y^{(i)})\text{cost}_0(\Theta^T x) = 0$

This reduces our cost function to:

$$J(\theta) = C\cdot 0 + \frac{1}{2}\sum_{j=1}^{n}\Theta_j^2$$
$$= \frac{1}{2}\sum_{j=1}^{n}\Theta_j^2$$

Recall the decision boundary from logistic regression (the line separating the positive and negative examples). In SVMs, the decision boundary has the special property that it is **as far away as possible** from both the positive and the negative examples.

The distance of the decision boundary to the nearest example is called the **margin**. Since SVMs maximize this margin, it is often called a *Large Margin Classifier*.

The SVM will separate the negative and positive examples by a **large margin**.

This large margin is only achieved when **C is very large**.

Data is **linearly separable** when a **straight line** can separate the positive and negative examples.

If we have **outlier** examples that we don't want to affect the decision boundary, then we can **reduce** C.

Increasing and decreasing C is similar to respectively decreasing and increasing λ, and can simplify our decision boundary.

Mathematics Behind Large Margin Classification (Optional)

Vector Inner Product

Say we have two vectors, u and v:

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

The **length of vector v** is denoted $||v||$, and it describes the line on a graph from origin (0,0) to (v_1, v_2).

The length of vector v can be calculated with $\sqrt{v_1^2 + v_2^2}$ by the Pythagorean theorem.

The **projection** of vector v onto vector u is found by taking a right angle from u to the end of v, creating a right triangle.

- p= length of projection of v onto the vector u.
- $u^T v = p \cdot ||u||$

Note that $u^T v = ||u|| \cdot ||v|| \cos\theta$ where θ is the angle between u and v. Also, $p = ||v||\cos\theta$. If you substitute p for $||v||\cos\theta$, you get $u^T v = p \cdot ||u||$.

So the product $u^T v$ is equal to the length of the projection times the length of vector u.

In our example, since u and v are vectors of the same length, $u^T v = v^T u$.

$u^T v = v^T u = p \cdot ||u|| = u_1 v_1 + u_2 v_2$

If the **angle** between the lines for v and u is **greater than 90 degrees**, then the projection p will be **negative**.

$$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\Theta_j^2$$
$$= \frac{1}{2}(\Theta_1^2 + \Theta_2^2 + \cdots + \Theta_n^2)$$
$$= \frac{1}{2}(\sqrt{\Theta_1^2 + \Theta_2^2 + \cdots + \Theta_n^2})^2$$
$$= \frac{1}{2}||\Theta||^2$$

We can use the same rules to rewrite $\Theta^T x^{(i)}$:

$\Theta^T x^{(i)} = p^{(i)} \cdot ||\Theta|| = \Theta_1 x_1^{(i)} + \Theta_2 x_2^{(i)} + \cdots + \Theta_n x_n^{(i)}$

So we now have a new **optimization objective** by substituting $p^{(i)} \cdot ||\Theta||$ in for $\Theta^T x^{(i)}$:

If y=1, we want $p^{(i)} \cdot ||\Theta|| \geq 1$

If y=0, we want $p^{(i)} \cdot ||\Theta|| \leq -1$

The reason this causes a "large margin" is because: the vector for Θ is perpendicular to the decision boundary. In order for our optimization objective (above) to hold true, we need the absolute value of our projections $p^{(i)}$ to be as large as possible.

If $\Theta_0 = 0$, then all our decision boundaries will intersect (0,0). If $\Theta_0 \neq 0$, the support vector machine will still find a large margin for the decision boundary.

Kernels I

Kernels allow us to make complex, non-linear classifiers using Support Vector Machines.

Given x, compute new feature depending on proximity to landmarks $l^{(1)}$, $l^{(2)}$, $l^{(3)}$.

To do this, we find the "similarity" of x and some landmark $l^{(i)}$:

$$f_i = similarity(x, l^{(i)}) = \exp(-\frac{||x - l^{(i)}||^2}{2\sigma^2})$$

This "similarity" function is called a **Gaussian Kernel**. It is a specific example of a kernel.

The similarity function can also be written as follows:

$$f_i = similarity(x, l^{(i)}) = \exp(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(i)})^2}{2\sigma^2})$$

There are a couple properties of the similarity function:

If $x \approx l^{(i)}$, then $f_i = \exp(-\frac{\approx 0^2}{2\sigma^2}) \approx 1$

If x is far from $l^{(i)}$, then $f_i = \exp(-\frac{(large\ number)^2}{2\sigma^2}) \approx 0$

In other words, if x and the landmark are close, then the similarity will be close to 1, and if x and the landmark are far away from each other, the similarity will be close to 0.

Each landmark gives us the features in our hypothesis:

$$\begin{matrix} l^{(1)} \rightarrow f_1 \\ l^{(2)} \rightarrow f_2 \\ l^{(3)} \rightarrow f_3 \\ \cdots \end{matrix}$$
$$h_\Theta(x) = \Theta_1 f_1 + \Theta_2 f_2 + \Theta_3 f_1 + \cdots$$

σ^2 is a parameter of the Gaussian Kernel, and it can be modified to increase or decrease the **drop-off** of our feature f_i. Combined with looking at the values inside Θ, we can choose these landmarks to get the general shape of the decision boundary.

Kernels II

One way to get the landmarks is to put them in the **exact same** locations as all the training examples. This gives us m landmarks, with one landmark per training example.

Given example x:

$f_1 = similarity(x, l^{(1)})$, $f_2 = similarity(x, l^{(2)})$, $f_3 = similarity(x, l^{(3)})$, and so on.

This gives us a "feature vector," $f_{(i)}$ of all our features for example $x_{(i)}$. We may also set $f_0 = 1$ to correspond with Θ_0. Thus given training example $x_{(i)}$:

$$x^{(i)} \rightarrow \left[f_1^{(i)} = similarity(x^{(i)}, l^{(1)}) f_2^{(i)} = similarity(x^{(i)}, l^{(2)}) \vdots f_m^{(i)} = similarity(x^{(i)}, l^{(m)})\right]$$

Now to get the parameters Θ we can use the SVM minimization algorithm but with $f^{(i)}$ substituted in for $x^{(i)}$:

$$\min_\Theta C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\Theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \Theta_j^2$$

Using kernels to generate f(i) is not exclusive to SVMs and may also be applied to logistic regression. However, because of computational optimizations on SVMs, kernels combined with SVMs is much faster than with other algorithms, so kernels are almost always found combined only with SVMs.

Choosing SVM Parameters

Choosing C (recall that $C = \frac{1}{\lambda}$

- If C is large, then we get higher variance/lower bias
- If C is small, then we get lower variance/higher bias

The other parameter we must choose is σ^2 from the Gaussian Kernel function:

With a large σ^2, the features fi vary more smoothly, causing higher bias and lower variance.

With a small σ^2, the features fi vary less smoothly, causing lower bias and higher variance.

Using An SVM

There are lots of good SVM libraries already written. A. Ng often uses 'liblinear' and 'libsvm'. In practical application, you should use one of these libraries rather than rewrite the functions.

In practical application, the choices you do need to make are:

- Choice of parameter C
- Choice of kernel (similarity function)
- No kernel ("linear" kernel) -- gives standard linear classifier
- Choose when n is large and when m is small
- Gaussian Kernel (above) -- need to choose σ^2
- Choose when n is small and m is large

The library may ask you to provide the kernel function.

Note: do perform feature scaling before using the Gaussian Kernel.

Note: not all similarity functions are valid kernels. They must satisfy "Mercer's Theorem" which guarantees that the SVM package's optimizations run correctly and do not diverge.

You want to train C and the parameters for the kernel function using the training and cross-validation datasets.

Multi-class Classification

Many SVM libraries have multi-class classification built-in.

You can use the *one-vs-all* method just like we did for logistic regression, where $y \in 1, 2, 3, \ldots, K$ with $\Theta^{(1)}, \Theta^{(2)}, \ldots, \Theta(K)$. We pick class i with the largest $(\Theta^{(i)})^T x$.

Logistic Regression vs. SVMs

If n is large (relative to m), then use logistic regression, or SVM without a kernel (the "linear kernel")

If n is small and m is intermediate, then use SVM with a Gaussian Kernel

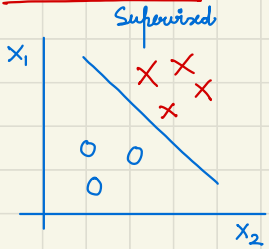
If n is small and m is large, then manually create/add more features, then use logistic regression or SVM without a kernel.

In the first case, we don't have enough examples to need a complicated polynomial hypothesis. In the second example, we have enough examples that we may need a complex non-linear hypothesis. In the last case, we want to increase our features so that logistic regression becomes applicable.

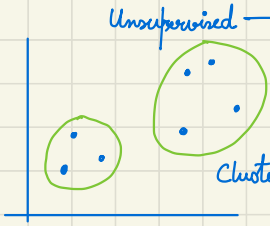
Note: a neural network is likely to work well for any of these situations, but may be slower to train.

UNSUPERVISED LEARNING

1. CLUSTERING



vs

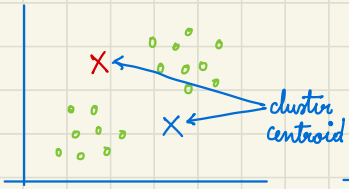


find me a pattern

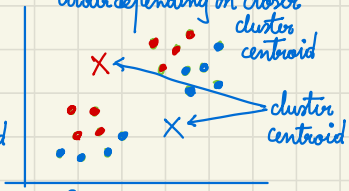
Clustering algorithm → market segmentation
→ social network analysis
→ organise computer clusters
→ galaxy formation

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

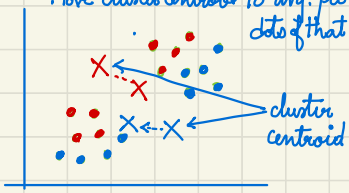
→ K-Means Algorithm



Colour depending on closer cluster centroid



Move cluster centroid to avg. loc of all dots of that colour



→ Randomly initialise K cluster centroids: $\mu_1, \ldots \mu_K \in \mathbb{R}^n$

Repeat { CLUSTER ASSIGNMENT → MOVE CENTROID }

CLUSTER ASSIGNMENT
for $i = 1$ to m
$c^{(i)} := \min_k \|x^{(i)} - \mu_k\|$

MOVE CENTROID
for $k = 1$ to K
$\mu_k :=$ avg of pts assigned to cluster k

Algorithm:
Input — K (# clusters)
— Training set: $\{x^{(1)}, x^{(2)}, \ldots x^{(m)}\}$ $x^{(i)} \in \mathbb{R}$ (drop $x_0 = 1$ convention)

→ K-Means for non-separated clusters
→ T-shirt sizing

$\min J(\ldots)$ wrt $c^{(1)}, \ldots c^{(m)}$
(holding $\mu_1, \ldots \mu_K$ fixed)

$\min J(\ldots)$ wrt $\mu_1, \ldots \mu_K$
(holding $c^{(1)} \ldots c^{(m)}$ fixed)

OPTIMISATION OBJECTIVE

K-means: $\{c^{(i)}, \mu_k\} \longrightarrow \mu_{c^{(i)}}$

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

loc. of cluster centroid to which $x^{(i)}$ has been assigned.

Distortion cost function

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

$$\min_{\substack{c^{(1)}, \dots c^{(m)} \\ \mu_1, \dots \mu_K}} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots \mu_K)$$

Random Initialisation

- $K < m$
- Pick random K training egs
- Set $\mu_1, \dots \mu_K$ = these K examples

Avoiding Local Optima

$\times$ 50–1000 $\longrightarrow$ Compute $J(c^{(1)}, \dots c^{(m)}, \mu_1, \dots \mu_K)$

pick clustering with lowest J

Choosing K:

→ Elbow method

→ Benchmark by downstream performance

eg: T-shirt size business

2. DIMENSIONALITY REDUCTION

MOTIVATIONS

→ **Data Compression**: Reduce data from 2D → 1D

$x^{(1)} \in \mathbb{R}^2 \longrightarrow z^{(1)} \in \mathbb{R}$

$\vdots$

$x^{(m)} \in \mathbb{R}^2 \longrightarrow z^{(m)} \in \mathbb{R}$

project onto reduced dimension

→ reduces disk space/storage

→ faster algorithm

k = # PC

→ **Data Visualisations**:

Dimensional reduction: $\{x^{(1)}, \dots x^{(m)}\} \longrightarrow \{z^{(1)}, \dots z^{(m)}\}$

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

$k \leq n$

→ $k = 2, 3$ (plot to visualise correlations)

→ **Analysis algorithm: PCA**

→ Problem formulation: find lower D surface to project data onto by minimising squared projected error

→ n-D → k-D $\{u^{(1)}, \dots u^{(k)}\}$: min Proj Error

↓ project onto linear subspace of these vectors

PCA ≠ Linear Regression

$x_1, x_2, \dots x_m$ — Unsupervised

$x \rightarrow y$ prediction — Supervised

→ Data preprocessing

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

Preprocessing: feature scaling / mean normalisation

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

$$x_j^{(i)} \rightarrow \frac{x_j - \mu_j}{s_j}$$

→ Algorithm: n-D → k-D

- Compute "covariance matrix"

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T \quad \text{— } n \times n$$

$n \times 1 \quad 1 \times n$

- Compute "eigenvectors" of Σ

$[U, S, V] = \text{svd}(\text{Sigma})$

or find linalg library for corresponding language

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

K; $n \rightarrow k$

$$U_{reduce} = z = \begin{bmatrix} | & & | \\ u^{(1)} & \dots & u^{(k)} \\ | & & | \end{bmatrix}^T x = \begin{bmatrix} — & (u^{(1)})^T & — \\ & \vdots & \\ — & (u^{(k)})^T & — \end{bmatrix} x$$

Summary:

→ Mean normalisation + Feature Scaling

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

PCA Algorithm

→ $[U, S, V] = \text{svd}(\text{Sigma})$

→ $U_{reduce} = U(:, 1:k)$ $\quad x \in \mathbb{R}^n$

→ $z = U_{reduce}^T * x$.

Compression: $z = U_{reduce}^T x$ $\quad n\text{-D} \rightarrow k\text{-D}$

Decompression: $k\text{-D} \rightarrow n\text{-D}$

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

Choosing k:

Avg sq. Proj Error: $\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2$

Tot. var. in data: $\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} \|^2$

Choose k: smallest value s.t.

$$\frac{\frac{1}{m}\sum \| x^{(i)} - x_{approx}^{(i)} \|^2}{\frac{1}{m}\sum \| x^{(i)} \|^2} \leq 0.01$$

99% is retained

Algorithm:

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

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

For given k: Check if

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

→ Increase k

→ Advice for applying PCA

→ Supervised learning speedup

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

Extract inputs:

Unlabeled training set: $x^{(1)}, \ldots x^{(m)} \in \mathbb{R}^{10000}$ (say)

PCA on training set → $z^{(1)}, \ldots z^{(m)} \in \mathbb{R}^{1000}$

mapping can be applied as well on $x_{cv}^{(i)}$ or $x_{test}^{(i)}$

→ New training set:

$\{(z^{(1)}, y^{(1)}), \ldots (z^{(m)}, y^{(m)})\}$

APPLICATION OF PCA

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

  ] Choose k by % of variance retained

- Visualization] k = 2 or 3

BAD USE OF PCA: Prevent Overfitting: $x^{(i)} \to z^{(i)}$ $k<n$ → less likely to overfit ~~X~~
→ might work OK. Instead use REGULARISATION
if 99% var retained it might throw away imp data

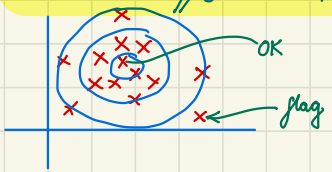
Design of ML System: Before implementing PCA, consider using original data $x^{(i)}$. If that doesn't work, consider dimensional Reduction

---

ANOMALY DETECTION

Problem: Given dataset $\{x^{(1)},..,x^{(m)}\}$
Is x_{test} anomalous?

→ Build model $p(x)$
If $p(x_{test}) < \varepsilon$ → Flag anomaly
$\geq \varepsilon$ → OK



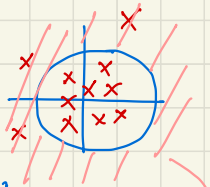
Example: FRAUD DETECTION
- $x^{(i)}$ = features of user i's activities
- Model $p(x)$ from data
- Id unusual users by checking $p(x) < \varepsilon$

MANUFACTURING
MONITERING COMPUTERS IN DATA CENTRES
- $x^{(i)}$: features of machine i
- $p(x) < \varepsilon$ to predict machines that might go down

ANOMALY DETECTION ALGORITHM WITH GAUSSIAN DIST

Training set: $\{x^{(1)}, \dots x^{(m)}\}$; $x \in \mathbb{R}^n$
Assume: $p(x) = p(x_1;\mu_1,\sigma_1^2)\,p(x_2;\mu_2,\sigma_2^2)\dots p(x_m;\mu_m,\sigma_m^2)$ [110]

Algorithm: $\{x^1,\dots x^m\}$
- Choose features x_i which might be indicative of anomalous examples
- Fit parameters: $\mu_1,\dots,\mu_n, \sigma_1^2,\dots\sigma_n^2$
$x^{(i)} \sim p(x^{(i)};\mu_j,\sigma_j^2)$ $\left[\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}\right]$ $\left[\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)}-\mu_j)^2\right]$
- Given new example x, compute $p(x)$: $p(x) = \prod_{j=1}^{n} p(x_j;\mu_j,\sigma_j^2) = \prod \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(x_j-\mu_j)^2}{2\sigma_j^2}}$



Anomaly if: $p(x) < \varepsilon$

→ Developing/Evaluating an Anomaly Detection System
- → w/ real # evaluation $y=1$ $y=0$
- → assume labelled data of anomalous/non-anomalous egs
- → training set: $\{x^{(1)}, \ldots, x^{(m)}\}$ assume non-anomalous
- → define CV and test sets

Motivating example: Aircraft engines

10000 good (normal) engines → 6000: Training ($y=0$) → $p(x) = p(x_1;\mu_1,\sigma_1^2) \ldots p(x_m;\mu_m,\sigma_m^2)$
- → 2000: CV + 10 anomalous ($y=1$)
- → 2000: Test + 10 anomalous ($y=1$)

20 flawed engines (non-anomalous) → Training

Algorithm:
- → Fit $p(x)$ on training set: $\{x^{(1)}, \ldots x^{(m)}\}$
- → On CV/test example, x, predict: $y = \begin{cases} 1 & \text{if } p(x) < \epsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \epsilon \text{ (normal)} \end{cases}$

↓ use CV to choose ϵ

Clarification accuracy not great ← skewed data set

instead use → Possible Eval. Metrics
- True +ve, Fase +ve
- True −ve, False−ve
- Precision/Recall
- F1 Score

ANOMALY DETECTION vs SUPERVISED LEARNING

→ very few $y=1$, +ve egs very large, $y=0$, −ve egs	→ large # of +ve or −ve egs
→ many different types of anom. ↳ hard for any algorithm to learn from +ve egs → future anom. might look nothing like anom. egs seen so far	→ enough +ve egs to get a sense of what +ve egs are like → generalisable to future
EXAMPLES Very few $y=1$	$y=0, y=1$ almost same
• Fraud detection • Manufacturing • monitoring machines in data clusters	• Email spam classification • Weather prediction • cancer classification

Choosing Features:
→ Non-Gaussian features → func^n transf^n to Gauss. fun^c

$x \longrightarrow \log(x)$ or $x^{0.05}$

→ Error analysis

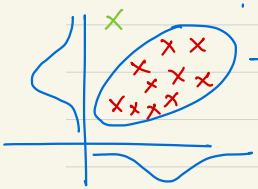
Wish: $p(x)$ large for normal egs, small for anom egs

Reality: comparable

For anomaly not being flagged, get inspiration for a new features

→ choose features which will take large or small values

ANOMALY DETECTION w/ MULTIVARIATE GAUSSIAN



→ doesn't assume covariance is diagonal with equal elements

→ multivariate Gaussian dist^n

$\sigma^2 \rightarrow \Sigma$

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

→ apply to anomaly detection algorithm

Parameters: μ, Σ

Parameter fitting: Given training set, $\{x^{(1)}, .., x^{(m)}\}$ $x \in \mathbb{R}^n$

Estimate: $\mu = \frac{1}{m}\sum x^{(i)}$

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

→ Fit model $p(x)$ by setting $\mu = .., \Sigma = ..$

→ Given new example x, compute $p(x)$

→ Flag anomaly if $p(x) < \epsilon$

→ Original model	vs.	→ Multivariate Gaussian
$p(x_1;\mu_1,\sigma_1^2) \times \cdots \times p(x_n;\mu_n,\sigma_n^2)$		$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}\lvert\Sigma\rvert^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$
Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values. → $x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}$		→ Automatically captures correlations between features
→ Computationally cheaper (alternatively, scales better to large n) $n=10{,}000$, $n=100{,}000$		Computationally more expensive → $\Sigma \in \mathbb{R}^{n\times n}$ Σ^{-1} → $n \approx \frac{n^2}{2}$
OK even if m (training set size) is small		Must have $m > n$ or else Σ is non-invertible. $m \geq 10n$ → $x_1 = x_4$, $x_3 = x_4 + x_5$

Andrew Ng

RECOMMENDER SYSTEMS → ALGORITHM

Example: Predicting movie ratings

n_u: #users

n_m: #movies

$r(i,j) = 1$ if user j has rated movie i

$y^{(i,j)}$ = rating given

→ Content-based recommender system

x_1 → romance $\in (0,1)$ $+ x_0 = 1$

x_2 → action $\in (0,1)$

For each user j, learn parameter $\theta^{(j)} \in \mathbb{R}^{n+1}$

Predict user j as rating movie i w/ $(\theta^{(j)})^T x^{(i)}$ stars

LINEAR REGRESSION for each user

Problems: given from data already rated what would a user have rated a movie not seen?

Problem Formulation: $r(i,j)$, $y^{(i,j)}$
$\theta^{(j)}$, $x^{(i)}$

$m^{(j)}$ = # movies rated by user j

To learn $\theta^{(j)}$: $\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}\right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (\theta_k^{(j)})^2$ — optimisation objective

[Linear Regression formulation]

To learn $\theta^{(1)}, \theta^{(2)}, \dots \theta^{(n_u)}$:

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

Gradient Descent Update:

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

$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left[\dots + \lambda \theta_k^{(j)}\right] \quad k \neq 0$

$\frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \dots \theta^{(n_u)})$

Basically linear Regression w/o $\frac{1}{m}$ terms

→ Collaborative Filtering:

learn features on its own

→ optimisation algorithm

Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, learn $x^{(1)} \dots x^{(n_m)}$

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

Collaborative filtering

[Given $\theta \rightarrow x$

[Given $x \rightarrow \theta$

Given $\theta \rightarrow x \rightarrow \theta \rightarrow x \dots$

Collaborating filtering

→ Collaborative filtering algorithm

Solve simultaneously

Collaborative filtering algorithm

1. Initialize $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values.
2. Minimize $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$ using gradient descent (or an advanced optimization algorithm). E.g. for every $j = 1, \dots, n_u, i = 1, \dots, n_m$:

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

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

$\frac{\partial}{\partial x_k^{(i)}} J(\dots)$

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

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

$x_0 = 1$, $x \in \mathbb{R}^n$, $\theta \in \mathbb{R}^n$, θ_0, θ_1, θ_n

Vectorised:

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

Predicted Ratings: $X\Theta^T$

Low Rank Matrix Factorisation

→ Finding Related Movies:

For pdt i, learn feature vector $x^{(i)} \in \mathbb{R}^n$

Relation btw movie $i \rightarrow j$

small $\|x^i - x^j\|$ → movies i & j are "similar"

+ MEAN NORMALISATION

LEARNING WITH LARGE DATASETS

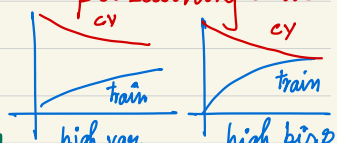
$m \sim O(10^6 - 10^7)$ (typical)

(Batch) Gradient descent: $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \dots$ ← sum over $O(10^6 - 10^7)$ sample for each iteration !!

→ use all m examples in each iteration

Perhaps select subset $\sim O(10^3)$

Sanity check → plot Learning curve



Increasing m ✗ ✓

→ Stochastic Gradient descent: → use 1 eg in each iteration

1. Randomly shuffle/reorder training examples
2. Repeat { (1-10x)
   for $i = 1, \dots, m$ {
   $\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad \forall j = 1, \dots, n$
   } }

Checking for convergence

→ compute cost $(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$ before updating θ using $(x^{(i)}, y^{(i)})$

→ every 1000 iter, plot cost ← averaged over 1000 egs

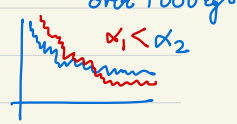
→ Mini-batch gradient descent → use b examples in each iteration

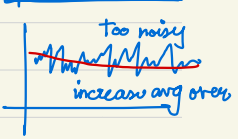
↓ mini batch size

> stochastic gradient descent ← iff Vectorization

for $i = 1, 11, \dots, 991$ {

if $b = 10$, $m = 1000$





Adapting Stochastic GD for

ONLINE LEARNING

→ unlimited data

→ can adapt to changing user preferences

→ 1 eg at a time

The PHOTO OCR Problem → computers to read text in images we take

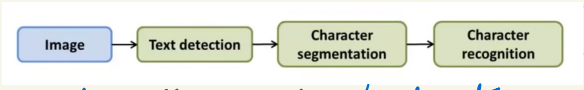
Given Photo, → Find where texts appear
→ Id/Read them

Optical Character Recognition

Uses → Search photos using text
→ camera for blind
→ car navigation

PHOTO OCR PIPELINE → text detection
→ character segmentation
→ character classification

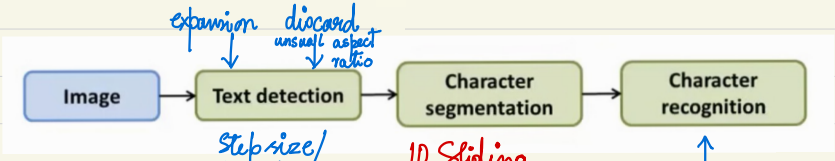
MACHINE LEARNING PIPELINE



A system with many stages/components several of which may use machine learning.

→ SLIDING WINDOWS CLASSIFIER →

Pedestrian detection → Text detection



Step size/stride param
2D Sliding window

1D Sliding window
Decide: whether split b/w 2 characters

Normal classifier algorithm

→ Synthetic data: Font library + random background ← unlimited supply of labelled data

→ Introducing distortions * CAVEAT: Distortions REPRESENTATIVE of egs in test set
→ not meaningless distortion

Speech Recognition
→ original audio + include distorted background
eg, bad cellphone
traffic background

1 training eg → many training egs

Discussion on getting more data

1. Make sure you have a low bias classifier before expending the effort. (Plot learning curves). E.g. keep increasing the number of features/number of hidden units in neural network until you have a low bias classifier.
2. "How much work would it be to get 10x as much data as we currently have?"
   - Artificial data synthesis
   - Collect/label it yourself
   - "Crowd source" (E.g. Amazon Mechanical Turk)

+ CEILING ANALYSIS
→ Giving right answers (also high variance/bias)
→ Id stage showing max. improvement
↓
work more on this stage

Summary: Main topics

Supervised Learning — $(x^{(i)}, y^{(i)})$ → Labelled data
- Linear regression, logistic regression, neural networks, SVMs

Unsupervised Learning — $x^{(i)}$ → unlabelled data
- K-means, PCA, Anomaly detection

Special applications/special topics
- Recommender systems, large scale machine learning. (including parallelized)

Advice on building a machine learning system
- Bias/variance, regularization; deciding what to work on next: evaluation of learning algorithms, learning curves, error analysis, ceiling analysis.

DEBUGGING

→ precision/recall
→ F1 score
→ training/CV/test set