Machine learning coussera Andrew Ng

Tom Mitchell 1998

A well posed learning problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T as measured by P improves with experience E.

Supervised learning → Regression (continuous output)
labelled data set. → classification (discrete valued output)

Unsupervised learning → clustering
unlabelled data set

Model representation. (Supervised learning)

$x^{(i)}$ input variables

$y^{(i)}$ output or target variable that we are trying to predict.

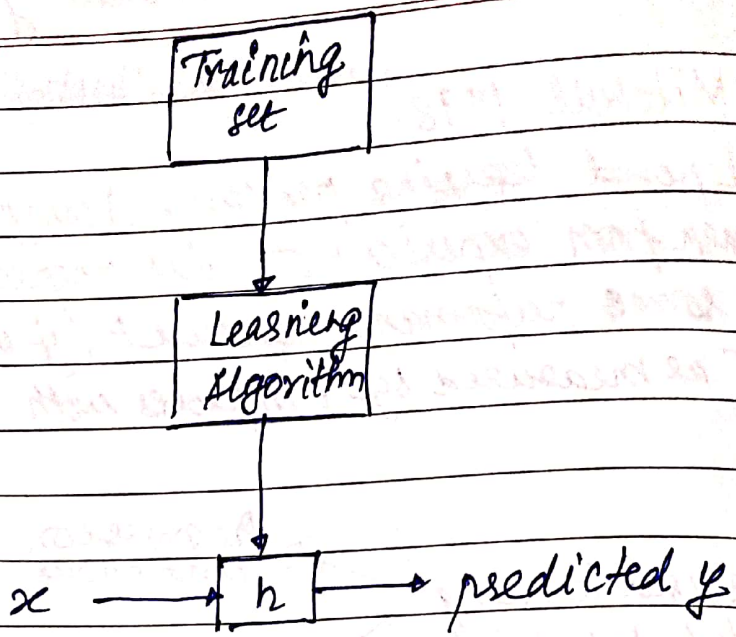
$(x^{(i)}, y^{(i)})$ is called a training example.

$(x^{(i)}, y^{(i)})$ $i = 1, \ldots m$ is called a training set

X space of input values.

Y space of output values.

To learn a function $h: X \to Y$ so that $h(x)$ is a "good" predictor for the corresponding value of y. (called hypothesis)



if y is continuous : regression
discrete : classification

~~Regression~~
example

Univariate Linear regression

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

cost function

θ_i s parameters
how to choose θ_i ?

minimise error $\sum_{i=1}^{n} (h_\theta(x) - y)^2$

minimise $\frac{1}{2m} \sum_{i=1}^{n} (h_\theta(x) - y)^2$
θ_0, θ_1

cost function $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Gradient descent

The gradient descent algorithm.
repeat until convergence
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

simultaneously update the parameters.

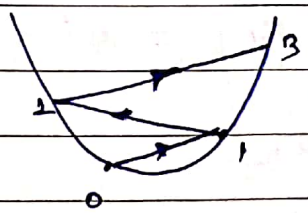
Correct
temp 0 := $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

temp 1 := $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

$\theta_0 :=$ temp 0
$\theta_1 :=$ temp 1

mathematically
$\vec{\theta} := \vec{\theta} - \alpha \vec{\nabla} J$

if α is too small gradient descent can be slow
if α is too large gradient descent can overshoot the min. It may fail to converge or even diverge



in case of univariate linear regression

$$J_\theta = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_i - y_i)^2$$

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{2m} \times \sum_{i=1}^{m}(\theta_0 + \theta_1 x_i - y_i)$$

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{2m} \times \sum_{i=1}^{m}(\theta_0 + \theta_1 x_i - y_i)\, x_i$$

Batch gradient descent.
this method looks at every example in the entire training set on every step

---

Multivariate linear regression

notation
- $x_j^{(i)}$ value of feature j in the i^{th} training eg
- $x^{(i)}$ input of the ith training eg.
- m the number of training eg.
- n the number of features.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 - \cdots + \theta_n x_n$$

$$h_\theta(x) = [\theta_0 \;\; \theta_1 \;\; \theta_2 \cdots \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

$$x_0^{(i)} = 1$$

hypothesis $\theta^T x$

Parameters θ n+1 vector

cost function $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

New algorithm

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

simultaneously update.

Feature scaling.

To speed up gradient descent

have input values roughly in same range

$x_i := \frac{x_i - \mu_i}{s_i}$ s_i range of values or standard dev

Debugging gradient descent

① Make a plot $J(\theta)$ vs number of iterations

If $J(\theta)$ increases then you probably need to decrease α.

② Automatic convergence ~~tests~~ Declare convergence if $J(\theta)$ decreases by less than ϵ in one iteration, where ϵ is some small value such as 10^{-3}. In practice it is difficult to choose this threshold value.

To summarise

If α is too small : slow convergence

If α is too large : may not decrease on every iteration and thus may not converge.

Features and Polynomial Regression

$h_\theta(x) = \theta_0 + \theta_1 x_1$

quadratic $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$

cubic $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

created new features $x_2 = x_1^2$

$x_3 = x_1^3$

scaling becomes impt

Normal equation

design matrix $X = \begin{bmatrix} x^{(1)T} \\ x^{(2)T} \\ \vdots \\ x^{(n)T} \end{bmatrix}$

solving $\nabla_\theta J = 0$

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

gradient descent	vs	Normal eqn
Need to choose α		No need to choose α
Needs many iterations		No need to iterate
$O(kn^2)$		$O(n^3)$
Works well when n large		slow if n is very large

Normal equation Non invertibility

X^TX is non invertible common causes

1. Redundant features linearly dependent.
2. Too many features ($m \leq n$) delete some features or use regularization.

Logistic regression (classification algorithm!)

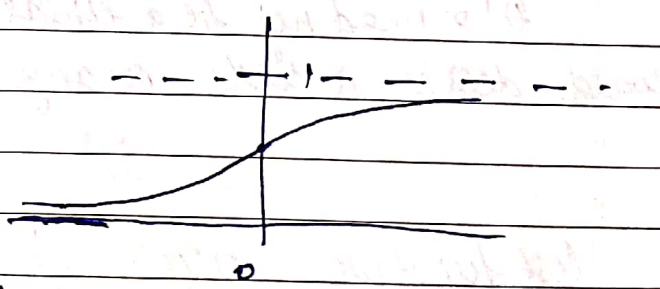
linear regression drawbacks

logistic regression model $0 \leq h_\theta(x) \leq 1$

$h_\theta(x) = g(\theta^T x)$

$g(z) = \frac{1}{1+e^{-z}}$

sigmoid func
logistic func



interpretation of hypothesis output

$h_\theta(x)$ = estimated probability that $y=1$ on inp x.

Ex: $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$

$h_\theta(x) = P(y=1 \mid x; \theta)$ prob that $y=1$ given x parameterized by θ.

logistic regression

$h_\theta(x) = g(\theta^T x) = P(y=1 | x;\theta)$

$g(z) = \frac{1}{1+e^{-z}}$

predict $y=1$ $h_\theta(x) \geq 0.5$

$y=0$ $h_\theta(x) < 0.5$

$g(z) \geq 0.5$

when $z \geq 0$

i.e $\theta^T x \geq 0$

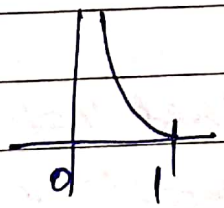
↑

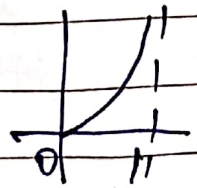
decision boundary

$\theta^T x$ need not be a linear function (polynomial regression could describe a circle or any other shape to fit our data.

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

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





guarantees convex

$$\text{cost} = (h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$

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

Vectorised implementation

$$h = g(X\theta)$$

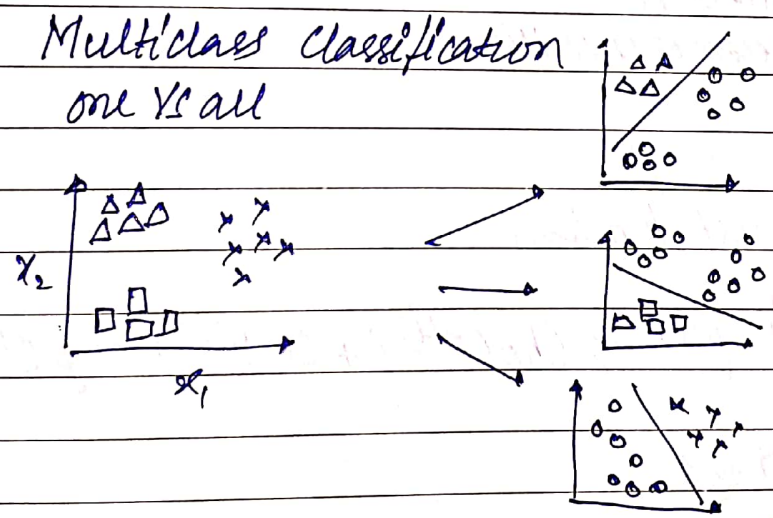
$$J(\theta) = \frac{-1}{m} \left(y^T \log(h) - (1-y)^T \log(1-h) \right)$$

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

Multiclass classification

one Vs all



To summasize

Train a logistic regression classifier for each class to predict the probability $y = i$

To make a new prediction pick the class that maximises $h_\theta(x)$

The problem of overfitting

regularized cost function

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

gradient

$\frac{\partial J}{\partial \theta_j} \quad \frac{1}{m}\left(\sum (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j\right)$

for $j \neq 0$

Normal equation

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

$L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$

if $m < n$ $X^T X$ is non invertible

but $X^T X + \lambda L$ becomes invertible

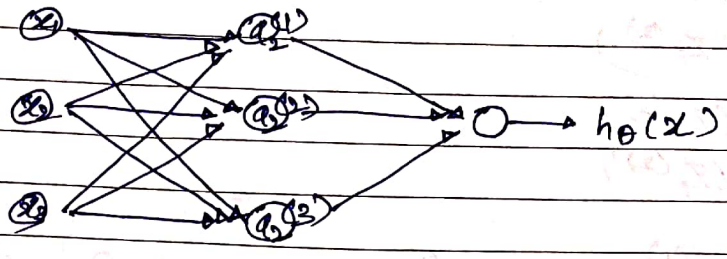
logestic regression regularized cost function.

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

$+ \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$

★ we dont regularize θ_0

Neural networks.



$a_i^{(j)}$ activation of unit i in layer j

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

If we had one hidden layer

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \longrightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} \longrightarrow h_\Theta(x)$$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j and s_{j+1} units in layer $j+1$ then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

↑ bias.

Vectorization.

$a_1^{(2)} = g(z_1^{(2)})$

$a_2^{(2)} = g(z_2^{(2)})$

$a_3^{(2)} = g(z_3^{(2)})$

$z_k^{(2)} = \Theta_{k,0}^{(1)} x_0 + \Theta_{k,1}^{(1)} x_1 \quad \cdots + \Theta_{k,n}^{(1)} x_n$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ \vdots \\ z_n^{(j)} \end{bmatrix}$$

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

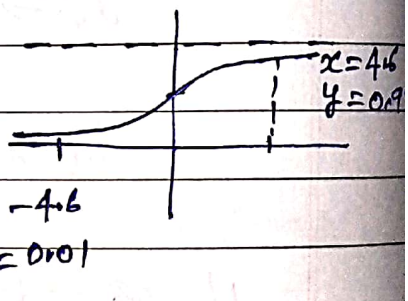
$a^{(j)} = g(z^{(j)})$

last step

$h_\Theta(x) = a^{(n)} = g(z^{(n)})$

Examples and intuitions I

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow [g(z^{(2)})] \rightarrow h_\Theta(x)$$



x_0 bias variable

$\Theta^{(1)} = [-30 \quad 20 \quad 20]$

$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

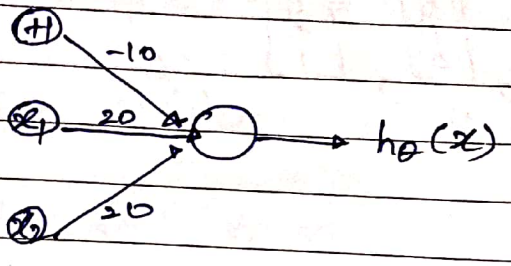
$x_1 = 0$ and $x_2 = 0$ $\quad g(-30) \approx 0$

$x_1 = 0$ and $x_2 = 1$ $\quad g(-10) \approx 0$

$x_1 = 1$ and $x_2 = 0$ $\quad g(-10) \approx 0$

$x_1 = 1$ and $x_2 = 1$ $\quad g(10) \approx 1$

OR function



x_1	x_2	$h_\theta(x)$
0	0	$g(-10)=0$
0	1	$g(10)=1$
1	0	$g(10)=1$
1	1	$g(30)=1$

AND

$\theta^{(1)} = [-30 \quad 20 \quad 20]$

NOR

$\theta^{(1)} = [10 \quad -20 \quad -20]$

OR

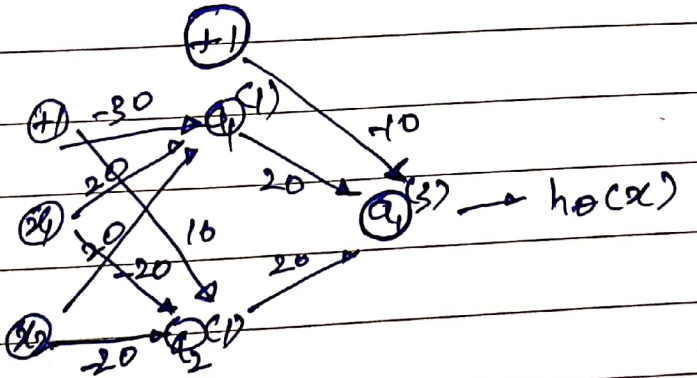
$\theta^{(1)} = [-10 \quad 20 \quad 20]$

XNOR

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \end{bmatrix} \rightarrow [a_1^{(3)}] \rightarrow h_\theta(x)$$

$$\theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$$

$$\theta^{(2)} = [-10 \quad 20 \quad 20]$$



multiclass classification using neural networks

If 4 objects $y^{(i)} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

↓ set of resulting classes

Cost function neural networks

L = total number of layers in the network.
S_l = number of units (not counting bias unit) in layer l.
K = number of output units/classes.

$h_\Theta(x)_k$ hypothesis that results in k^{th} output

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left((h_\Theta(x^{(i)}))_k\right) + (1-y_k^{(i)}) \log\left(1-h_\Theta(x^{(i)})\right)_k$$

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} (\Theta_{j,i}^{(l)})^2$$

↓ excluding the bias unit

Backpropagation

Training set $\{(x^{(i)}, y^{(i)})\}$ $i = 1 \cdots m$

set $\Delta_{ij}^{(l)} = 0$ for all l, i, j

For $i = 1$ to m

set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3 \ldots L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

compute $\delta^{(L-1)}, \delta^{(L-2)} \cdots \delta^{(2)}$

$$\delta^{(l)} = (\theta^{(l)})^T \delta^{(l+1)} * g'(z^{(l)}) \longrightarrow a^{(l)} * (1 - a^{(l)})$$

$$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

Vectorised $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)} \quad j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad j = 0$$

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)}$$

Implementation

1. Unrolling parameters
2. gradient checking
3. random initialisation

Evaluating a hypothesis

methods of troubleshooting

1. getting more training examples.
2. Trying smaller sets of features.
3. Trying additional features.
4. Trying polynomial features.
5. increasing or decreasing λ.

dataset

training set — test set

1. learn $\vec{\theta}$ and minimize $J_{train}(\vec{\theta})$ using the training set.

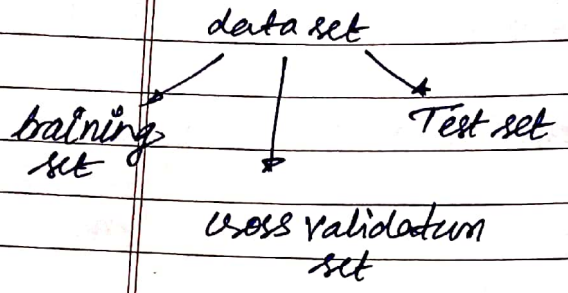
2. Compute the test set error $J_{test}(\vec{\theta})$

misclassification error

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

$$\text{Test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\theta(x_{test}^{(i)}), y_{test}^{(i)})$$

Model selection and train/validation/test sets.

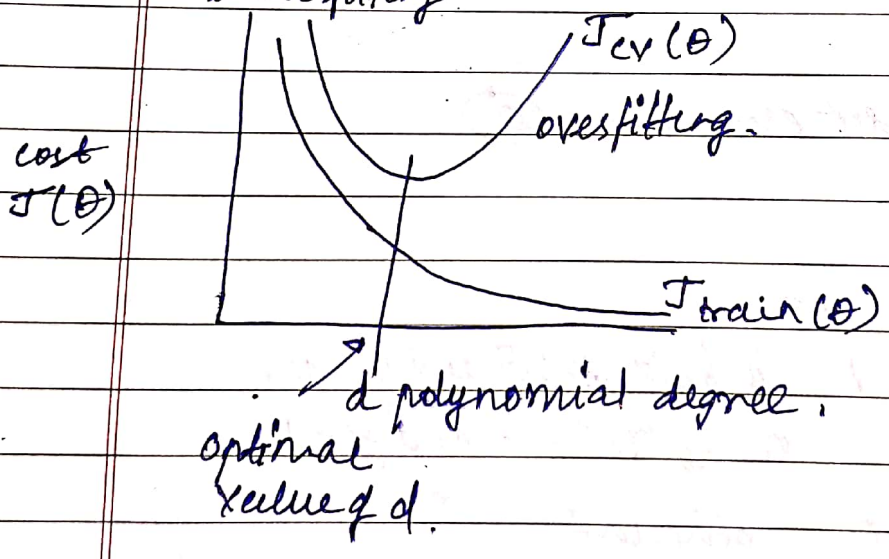


① Optimise the parameters in θ using the training set for each polynomial degree.

② Find the polynomial degree with least error using the cross validation ~~test~~ set.

③ Estimate the generalization error using the test set with $J_{test}(\theta^{(d)})$ d = theta from polynomial with lowest error

Diagnosing Bias vs Variance



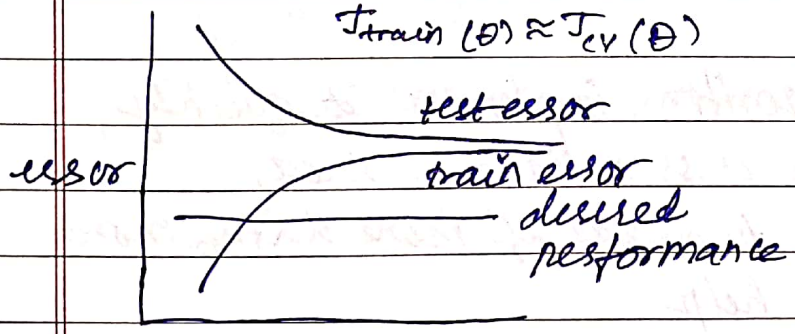
(i) High bias (underfitting) both $J_{train}(\theta)$ and $J_{cv}(\theta)$ will be high also $J_{cv}(\theta) \simeq J_{train}(\theta)$

(ii) High variance (overfitting) $J_{train}(\theta)$ will be low and $J_{cv}(\theta)$ will be much greater than $J_{train}(\theta)$.

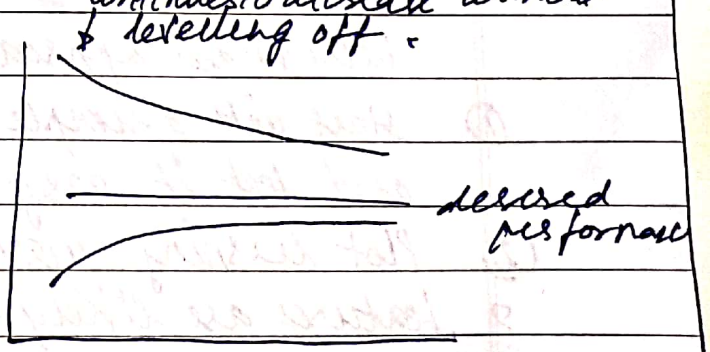
Regularization and Bias/Variance

1. Create a list of λs $\lambda = 0 \quad 0.01 \quad 0.02 \quad 0.04 \quad \dots \quad 10.24$
2. Create a set of models with different degrees or any other variants.
3. Iterate through λs and for each λ go through all the models to learn some θ.
4. Compute the cross validation error using the learned θ (computed with λ) on the $J_{cv}(\theta)$ without regularization
5. Select the best combo that produces the lowest error on the cross validation set.
6. Using the best combo θ and λ, apply it on $J_{test}(\theta)$ to see if it has a good generalization of the problem

Learning curves



high bias.
more training data will by itself not help much.



high variance
getting more data is likely to help.

① getting more training examples (fixes high variance)

② Trying smaller set of features (fixes high variance)

③ Adding features (Fixes high bias)

④ adding polynomial features (fixes high bias)

⑤ ↓ λ (fixes high bias)

⑥ ↑ λ (fixes high variance)

Machine Learning system design.

Prioritizing on what to work on.

① collect lots of data.
② Develop sofisticated features.
③ Develop algorithms to process your input in different ways.

Recomended approach

① start with a simple algorithm, implement it quickly, and test it easly on cross validation data.
② Plot learning curves to decide if more data, more features are likely to help.
③ Manually examine the errors on the examples in the cross validation set and try to spot a trend where most of the errors were made.

We should try new things, get a numerical value of our error rate, and based on our result decide whether to keep the new feature or not.

Handling skewed ~~classes~~ data

Precision / Recall

		Actual class 1	Actual class 0
Predicted class	1	True positive	False positive
	0	False negative	True negative

$$\text{Precision} = \frac{\text{True positives}}{\#\text{predicted positive}}$$

$$\text{Recall} = \frac{\text{True positives}}{\#\text{Actual positives}}$$

Trading off precision and recall.

threshold $= 0.7$

1 if $h_\theta(x) \geq 0.7$

0 if $h_\theta(x) < 0.7$

higher precision lower recall.

threshold $= 0.3$

lower precision higher recall.

$$F_{score} = \frac{2PR}{P+R}$$

Useful test: Given the input x, can a human expert confidently predict y?

Large data rationale

Use a learning algorithm with many parameters
low bias algo
$J_{train}(\theta)$ will be small.

Use a very large training set (unlikely to overfit)
$J_{train}(\theta) \approx J_{test}(\theta)$

$J_{test}(\theta)$ will be small.

Unsupervised learning

K means algorithm

Input
- K numbers of clusters
- Training set $\{x^{(1)}, x^{(2)}, x^{(3)} \ldots x^{(m)}\}$

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

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

Repeat {

cluster assign step (holding μ const, min wrt c)

for $i = 1$ to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$ $\quad \min_k \| x^{(i)} - \mu_k \|$

move centroid step (holding c const, min wrt μ)

for $k = 1$ to K

μ_k := average (mean) of points assigned k^{th} cluster

}

optimisation objective

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

distortion function

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

Random initialization
Should have $K < m$
Randomly pick K training examples
Set $\mu_1, \ldots \mu_K$ equal to these K examples

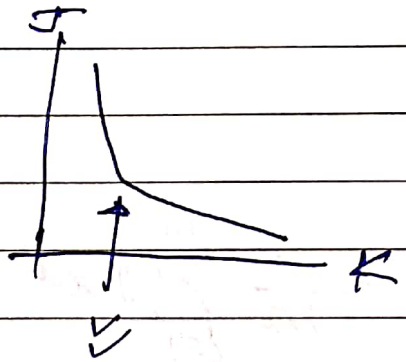
To avoid local optima

for $i = 1$ to 100 {
    Randomly initialize K-means
    Run K-means get $c^{(1)} \ldots c^{(m)}$ $\mu_1 \ldots \mu_K$
    Compute cost function (distortion)
}

pick clustering that gave lowest $J(c, \mu)$

Choosing number of clusters

① elbow method.



② evaluate K means based on a metric for how well it performs for that later purpose.

Dimentionality Reduction & PCA.

Principal component Analysis PCA.

Data preprocessing

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

$$x_j^{(i)} := \frac{x_j^{(i)} - \mu_j}{s_j}$$ ← mean normalization, ← feature scaling

PCA algorithm

Compute covariance matrix

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T \quad \frac{1}{m} * X' * X$$

Compute eigenvectors of Σ

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

reconstruction

$$x_{approx}^{(i)} = Ureduce * z^{(i)}$$

choosing k number of principal components

Algorithm

Try PCA with $k = 1$

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

Check if $\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} = 1 - \frac{\sum_{1}^{k} S_{ii}}{\sum_{1}^{n} S_{ii}} \leq 0.01$

99% variance retained.

Supervised learning speedup

$(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$ ---

PCA from $x^{(i)}$ to $z^{(i)}$

$(z^{(1)}, y^{(1)})$ $(z^{(2)}, y^{(2)})$ ---

Note mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on the training set. This mapping can be applied as well to $x_{cv}^{(i)}$ $x_{test}^{(i)}$.

try running algo w/o PCA.

Anomaly detection

Data set $\{x^{(1)}, x^{(2)}, \ldots x^{(m)}\}$
is x_{test} anomalous?

model $p(x)$ $\quad p(x_{test}) < \varepsilon$ flag anomaly
$\quad p(x_{test}) \geq \varepsilon$ ok

Gaussian distribution
$x \sim N(\mu, \sigma^2)$ $\quad \sigma$ standard deviation

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

If each feature $\in \mathbb{R}^n$

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

algorithm

① choose features that you think might be indicative of anamdous examples.

② fit parameters $\mu_1 \ldots \mu_n \quad \sigma_1^2 \ldots \sigma_n^2$

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

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

③ given new example compute $p(x)$

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

Anomaly if $p(x) < \varepsilon$

Algorithm evaluation

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

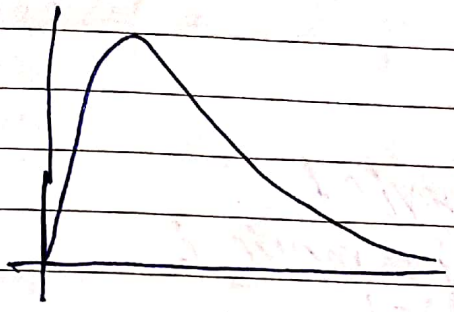
use precision and recall and F_1 score to evaluate

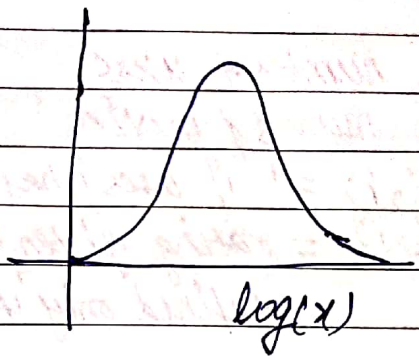
Can also use cross validation test to choose ϵ.

Anomaly detection vs Supervised learning

Anomaly detection	Supervised learning
① Very small number of +ve examples ($y=1$) (0-20 is common)	Large number of +ve and -ve examples.
② Large number of -ve examples	
③ Many different types of anomalities. Hard for any algorithm to learn from +ve examples what the anomality looks like. future anomalities look ~~very~~ nothing like the ones were seen so far.	Enough +ve examples for algorithm to get a sense of what +ve eg are like. future +ve egs are likely to be similar to ones in training set.
eg Fraud detection, manufacturing, monitoring machines in data center	email spam classification, weather prediction (sunny/rainy), cancer classification.

choosing what features to use.





$\log(x+c)$

$x^{1/c}$

choose features that might take on unusually large or small values in the event of an anomaly

Multivariate gaussian distribution

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

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

original model	multivariate gaussian
manually create features to capture anomalities where x_1, x_2 take unusual combinations of values.	Automatically captures correlations b/w features
Computationally cheaper scales better to large n	computationally more expensive $m > n$

Recomender systems

n_u numbes of users
n_m numbes of movies
$r(i,j) = 1$ if user j has rated movie i.
$y^{(i,j)}$ = rating given by user j to movie i
defined only if $r(i,j)=1$.

content based recommendes systems
For each user j learn a parameter $\theta^{(j)} \in \mathbb{R}^{n+1}$ predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars.

Problem formulation

$\theta^{(j)}$ parameter vector for user j.
$x^{(i)}$ feature vector for movie i
For user j, movie i predicted rating $(\theta^{(j)})^T (x^{(i)})$

To learn $\theta^{(j)}$

optimization objective:

To learn $\theta^{(j)}$ (parameter for user j):

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

To learn $\theta^{(j)}$ $\forall j = 1, 2 \cdots n_u$

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

gradient descent update

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} \quad \text{for } k=0$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \quad k \neq 0$$

Collaborative filtering

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(i)}$:

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

given $\theta^{(1)}, \ldots \theta^{(n_u)}$, to learn $x^{(1)}, \ldots, x^{(n_m)}$:

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

given $x^{(1)}, \ldots x^{(n_m)}$ (and movie ratings)
can estimate $\theta^{(1)} \ldots \theta^{(n_u)}$

given $\theta^{(1)} \ldots \theta^{(n_u)}$
can estimate $x^{(1)} \ldots x^{(n_m)}$

Collaborative filtering optimization objective

Minimizing $x^{(1)}, \ldots x^{(n_m)}$ and $\theta^{(1)}, \ldots \theta^{(n_u)}$ simultaneously

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

Collaborative filtering algorithm

1) Initialize $x^{(1)}, \ldots, x^{(n_m)}$ $\theta^{(1)}, \ldots \theta^{(n_u)}$ to small random values

2) Minimise $J(x^{(1)}, \ldots x^{(n_m)}$ & $\theta^{(1)}, \ldots \theta^{(n_u)})$ by gradient descent

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

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

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

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

$$\Theta = \begin{bmatrix} (\theta^{(1)})^T \\ \vdots \\ (\theta^{(n_u)})^T \end{bmatrix}$$

$X \Theta^T$ predicted ratings

Mean normalization

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

Large scale machine learning.

Stochastic gradient descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{m} \sum cost(\theta, (x^{(i)}, y^{(i)}))$$

① Randomly shuffle data set

② Repeat {

for $i = 1, \ldots, m$ {

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

(for $j = 0, \ldots n$) simultaneous update

}

}

Mini batch gradient descent

Batch gradient descent: Use all m examples in each iteration

Stochastic gradient descent: Use 1 example in each iteration

Mini-Batch gradient descent: Use b examples in each iter.

Say $b = 10$, $m = 1000$

Repeat {

for $i = 1, 11, 21, \ldots 991$ {

$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$

(for every $j = 0 \ldots n$)

}

}

checking for convergence

① Batch gradient descent:

→ Plot $J_{train}(\theta)$ as a func of number of iterations of gradient descent.

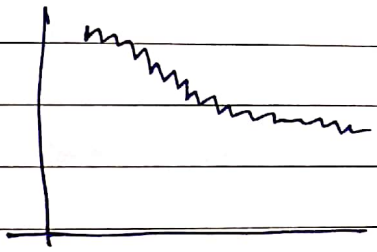
→ $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

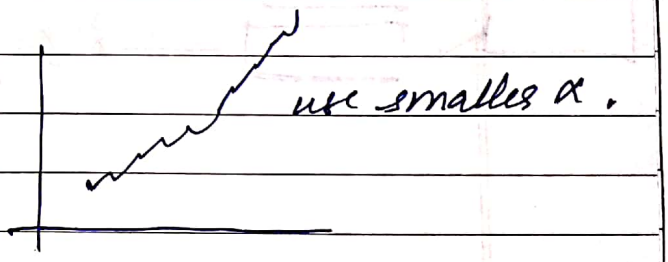
② Stochastic gradient descent

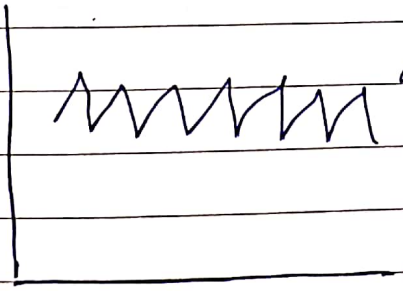
→ $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$

→ During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$

→ Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by the algo.







→ We can slowly decrease α over time if we want θ to converge

$\alpha = \frac{const1}{IterationNumber + const2}$

Online learning

Repeat forever {
get (x, y)
Update θ using (x, y) :
$\theta_j := \theta_j - \alpha (h_\theta(x) - y) x_j \quad j = 0 \dots n$
}

Can adapt to changing user preferences.

Map reduce

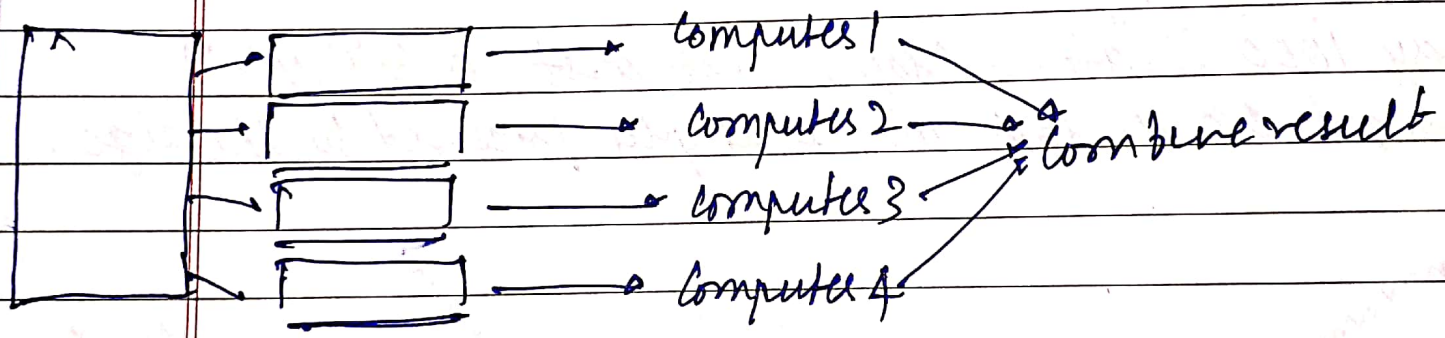
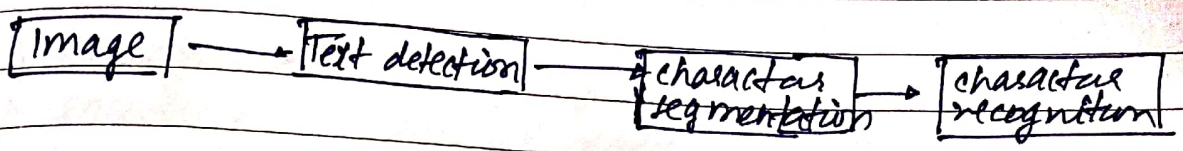


Photo OCR problem



On getting more data

1) Make sure you have a low bias classifier before expending the effort. (Plot learning curves) Keep increasing # features / number of hidden units in a 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 data yourself
→ 'Crowd source' Amazon Mechanical Turk.

ceiling analysis.

Component	Accuracy	
Overall system	72%	
Text detection	89%	↓ 17%
characters segmentation	90%	↓ 1%
Characters recognition	100%	↓ 10%