Machine learning

— coursera

Andrew Ng

Start off from Multivariate linear regression

$x_j^{(i)}$ = value of feature j in ith training example

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

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

Gradient descent: Repeat{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

Therefore cost function will be:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=0}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \quad (m \text{ examples})$$

Simultaneously update θ_j ($j = 0, 1 \cdots n$)

We can speed up gradient descent by (it will take less iterations)

feature scaling and mean normalization.

$$x_i := \frac{x_i - u_i}{s_i}$$

if α is sufficient small, $J(\theta)$ will decrease every iteration.

Polynomial regression:

$h_\theta(x) = \theta_0 + \theta_1 x_1$ let $x_2 = x_1^2$

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

if choose feature this way, feature scaling becomes really important.

Normal equation (without using Gradient Descent)

Gradient descent	Normal equation
choose α	No need to choose α
need many iterations	No need to iterate
$O(kn^2)$	$O(n^3)$
Works well if n is large	slow if very large.

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

x_0	x_1	x_2	x_3	y
⋮	Some number			some number

prove:

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

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

$$= X^T X\theta - X^T \vec{y}$$

or can understand as:

$X\theta = Y$

$X^T(X\theta - Y) = 0$

order make difference

What if $X^T X$ is invertible:

- Redundant feature
- Delete some feature. or use regularization.

Octave

xor(1,0) ans=1

% to comment

disp( ) (= print)

sprintf('2 decimals: %0.2f', a)

format long

A = [12; 34; 56]

V = [1;2;3]

v = 1:0.1:2

V = 1:6

ones(2,3)
zeros(3.3)
rand(3,3)
sqrt(a)
hist( )
eye(4)
help eye.

Size(A)
size(A,2) the second number in size
length(a) ⇒ vector

pwd
cd
ls
load file
load('file')
who
whos more detail command
clear <variable>
<variable>(1:10) index using ( )
save hello.mat V;
save hello.txt V -ascii

A([13],:)

A = [A, [column]]

A(:) %put all element together

C = [A B] == C=[A, B]

C = [A; B]

A*C dot

A.*C element wise

A.^2 square

1./V every element

log(e)
V(e)

abs(a)

V+1

A' A transpose

val = max(a)

[val, ind] = max(a) val and index of a.

a<3. ⇒ [1,0,1,1] 1 for True 0 for False

find(a<3) all the index of value

A = magic(3)

[r,c] = find(A>=7)

A(2,3) = 7

sum(a)
ceil(a)
floor(a)
prod(a)
max(rand(3), rand(3)) ⇒ two sigms
max(A,[],1) first dimension max } A= magic(3)
max(max(A))

sum(A,1) [.....]

sum(A,2) ⋮

A.*eye(9)

flipud(eye(9))

pinv(A) % inverse of A.

plotting

```
plot(x,y)
hold on;
plot(x,y2)
xlabel('...')
legend('sin','cos')
title('my plot')
save file:
print -dpng 'myplot.png'
close
figure(1); plot(t,y1);
subplot(1,2,1);
subplot(1,2,2);
axis([xmin xmax ymin ymax])
imagesc(A)
imagesc(A), colorbar, colormap gray;
```

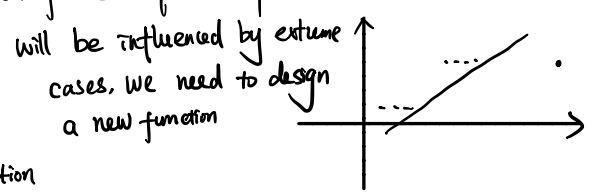
```
for i=1:10,
    v(i) = 2^i;
end;

break
continue,

i=5
while i<=5,
   v(i)=100;
   i=i+1
end;

if, elseif, else end;

function y= squareThisNumber(x)
       y= x^2;

function [y1,y2] = y2(x)
          ....

function J = costFunctionJ(X,y,theta)

  m=size(X,1)
  predictions = X*theta
  sqrErrors = (predictions-y).^2
  J = 1/(2*m) sum(sqrErrors)
```

a file with .m

Vectorization

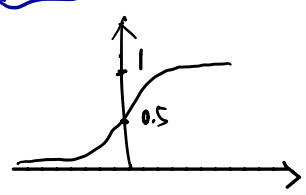
Classification:

Do not usually use linear regression for classification problem.



logistic regression:

$$\begin{cases} h_\theta(x) = g(\theta^T x) \\ g(z) = \frac{1}{1+e^{-z}} \end{cases}$$ Sigmoid function / Logistic function



$h_\theta(x) = P(y=1 \mid x;\theta)$ probablity $y=1$, given x parameteized by θ

$y=1$ if $h_\theta(x) \geq 0.5 \Rightarrow \theta^T x \geq 0$

$y=0$ if $h_\theta(x) < 0.5 \Rightarrow \theta^T x < 0$

Decision boundary: separate two cases, $h_\theta(x) = 0.5$

it is a property of parameter instead of a property of a dataset

Non-linear decision boundary:

$h_\theta x = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3^2 + \theta_4 x^2)$

Cost function

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

convex function

if $y=1$, but predict $y=0$, we give it big penalty.

$Cost(h_\theta(x), y) = -(1-y)\log(1-h_\theta(x)) - y\log(h_\theta(x))$

(principle of maximum likelihood optimization to get cost function)

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

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

update θ

$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}(J(\theta)) \Rightarrow$ compute by chain rule.

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

Vectorized implementation:

$$\theta := \theta - \frac{\alpha}{m} x^T(g(x\theta) - \vec{y})$$

other optimization algorithm:

- Gradient descent
- Conjugate descent — No need α, faster than gradient descent
- BFGS
- L-BFGS

```
function [jval, gradient] = costFunction(theta)
    jval = (theta(1)-5)^2 + (theta(2)-5)^2
    gradient = zeros(1,2)
    gradient(1) = 2*(theta(1)-5)
    gradient(2) = 2*(theta(2)-5)

options = optimset('GradObj','on','MaxIter','100')
initialTheta = zeros(2,1)
[optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options)
```

multiclass classification: one vs all method

$h_\theta^{(i)}(x) = P(y=i \mid x;\theta)$

$\max_i h_\theta^{(i)}(x)$ to be the correct category

Overfitting:

assumption causing bias

underfit high bias

overfitting low bias

Address overfitting:

1. Reduce number of features
   - Manually select which feature to keep.
   - Model selection algo
2. regularization
   - keep all features reduce value/magnitude of θ_j
   - works well if we have a lot of features each contribute a bit to y.

Small values θ_1 θ_2
- 'simpler' hypothesis
- less prone to over fitting

$$J(\theta)=\frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2+\lambda\sum_{j=1}^{n}\theta_j^2\right] \quad (R_2)$$

$\min_\theta J(\theta)$

conventionally, usually start at 1

Gradient descent

Repeat {

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)})-y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j\underbrace{(1-\alpha\frac{\lambda}{m})}_{\approx 0.99} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}$$

shrinking θ_j a little bit.

Normal equation regularization

$$\underbrace{(X^TX+\lambda\begin{bmatrix}0&&&\\&1&&\\&&\ddots&\\&&&1\end{bmatrix})^{-1}}_{\text{optional if } \lambda>0,\ \text{is invertible}}X^Ty \qquad \frac{\partial}{\partial\theta_j}J(\theta)=0$$

Regularized logistic regression:

Repeat {

$$\theta_0 := \theta_0 - \frac{\alpha}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)})-y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j(1-\alpha\frac{\lambda}{m}) + \alpha\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}$$

(Simultaneously update all θ_j, $j=1,\dots,n$)

(same as above)

Cost function:

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

function [jval, gradient] = costFunction(theta)

jval = [code to compute $J(\theta)$]

gradient(1) = [code to compute $\frac{\partial}{\partial\theta_0}J(\theta)$]

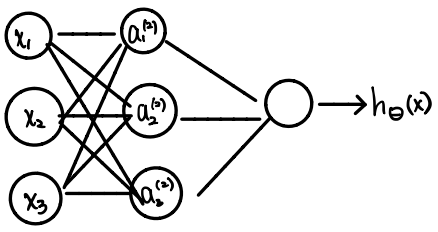
$$\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_0^{(i)}$$

gradient(2) = [code to compute $\frac{\partial}{\partial\theta_0}J(\theta)$]

⋮

gradient(n+1) = [....]

$$\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_n^{(i)}+\frac{\lambda}{m}\theta_{n+1}$$



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

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

if network has s_j units in layer j, s_{j+1} units in layer $j+1$.

then $\Theta^{(j)}$ will be dimension $s_{j+1} \times (s_j + 1)$

Forward Propagation:

$z^{(2)} = \Theta^{(1)} a^{(1)}$ (add $a_0^{(1)} = 1$)

$a^{(2)} = g(z^{(2)})$ ($g()$ logistic)

Θ_{ij} from j to i

layer n layer $n+1$

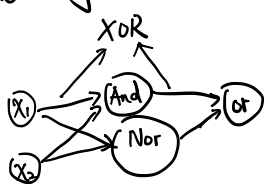
$a_1^{(2)} = g(\Theta_{10}^{(1)} a_0^{(1)} + \Theta_{11}^{(1)} a_1^{(1)} + \Theta_{12}^{(1)} a_2^{(1)} + \Theta_{13}^{(1)} a_3^{(1)})$

for logical function:

$h_\Theta(x) = g(-30 + 20x + 20y)$ AND

$h_\Theta(x) = g(-10 + 20x + 20y)$ OR

$h_\Theta(x) = g(10 - 20x)$ NOT



Neural network (classification)

L = total layers s_l = no. units (not counting bias unit) in layer l

k = number of output units classes $h_\Theta(x)_k$: hypothesis result in k^{th} output.

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

(by convention, do not add in x_0)

for multiclass classification.

back propagation:

$\delta_j^{(l)}$ = "error" of node j in layer l.

$\delta_j^{(4)} = a_j^{(4)} - y_j$ $\quad \delta^{(4)} = a^{(4)} - y$

$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$ ← $a^{(3)} .* (1 - a^{(3)})$

$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$

prove ignored:

if ignore regularization term:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

Back propagation algorithm:

Traning set $\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$

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

For $i = 1$ to m

- set $a^{(1)} = x^{(i)}$
- Perform FP to compute $a^{(l)}$ for $l = 2, 3 \cdots L$
- Using $y^{(i)}$, Compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
- Compute $\delta^{(L-1)}, \delta^{(L-2)}, \cdots \delta^{(2)}$
- $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + [a_j^{(l)}]\delta_i^{(l+1)}$

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

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

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

(FP using $x^{(1)}$ followed by BP using $y^{(1)}$. FP using $x^{(2)}$ followed by BP using $y^{(2)}$.)

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

The capital delta matrix D is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative. Thus we get $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$

Forward propagation:

$a^{(1)} = x$

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

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

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

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

$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* g'(z^{(l)})$$

using chain rule.

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

Neural network in practise (In Octave)

Example

$\Theta^{(1)} \in \mathbb{R}^{10 \times 11}$, $\Theta^{(2)} \in \mathbb{R}^{10 \times 11}$, $\Theta^{(3)} \in \mathbb{R}^{1 \times 11}$

$D^{(1)} \in \mathbb{R}^{10 \times 11}$, $D^{(2)} \in \mathbb{R}^{10 \times 11}$, $D^{(3)} \in \mathbb{R}^{1 \times 11}$

```
thetaVec = [Theta1(:); Theta2(:); Theta3(:)];
DVec = [D1(:); D2(:); D3(:)];

unroll:
Theta1 = reshape(thetaVec(1:110), 10, 11);
Theta2 = reshape(thetaVec(111:220), 10, 11);
Theta3 = reshape(thetaVec(221:231), 1, 11);
```

Learning Algo:

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, backward prop to compute D(1) D(2) D(3) and J(Θ).
    Unroll D(1), D(2), D(3) to get gradientVec.
```

Numeric estimation of gradients:

$$gradApprox = \frac{(J(\theta+\varepsilon) - J(\theta-\varepsilon))}{2\varepsilon}$$

Remember to do gradient checking.

```
epsilon = 1e-4.
for i = 1:n
    thetaPlus = theta;
    thetaPlus(i) += ε;
    thetaMinus = theta;
    thetaMinus -= ε;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*ε)
end;
```

Neural network trainning:

Random initialization:

PROBLEM:

Zero initialization:

$a_1^{(2)}$ $a_2^{(2)}$, Also $\delta_1^{(2)} = \delta_2^{(2)}$

$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta)$

After update, parameters are still the same.

→ Therefore, Pick theta value $\in (-\varepsilon, \varepsilon)$.

(e.g. rand(a,b) * (2* init_epsilon) - init_epsilon;

Training a neural network

1. randomly initialize weights
2. implements forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. implement code to compute $J(\Theta)$
4. implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{j,k}^{(l)}} J(\Theta)$

→ for i = 1:m

Perform forward and backward prop using $(x^{(i)}, y^{(i)})$

(Back prop algorithm)

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} (J(\Theta))$ computed using backprop is numeric greatest of $J(\Theta)$

Then disable checking code.

6. Use gradient descent or advanced optimization method with backprop to try to minimize $J(\Theta)$. (not convex)

Week 6.

Evaluating an algorithm:

training set 70% test set 30%

m_{test} = No. of test set cases

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

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1-y_{test}^{(i)}) \log\left(1-h_\theta(x^{(i)})\right)$$ classification

Misclassification error

$$err(h_\theta(x), y) = \begin{cases} 1 \\ 0 \end{cases}$$ if $h_\theta(x) \geq 0.5$ $y=0$ OR $h_\theta(x) \leq 0.5$ $y=1$

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

Choose model selection on valid set.

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

others is same as others.

For this example:

1. optimize parameter Θ using training set.
2. find polynomial degree d with least error using cross validation set.
3. estimate generalization error using test set with $J_{test}(\Theta^d)$

Bias (underfit):

$J_{train}(\theta)$ will be high

$J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit)

$J_{train}(\theta)$ will be low

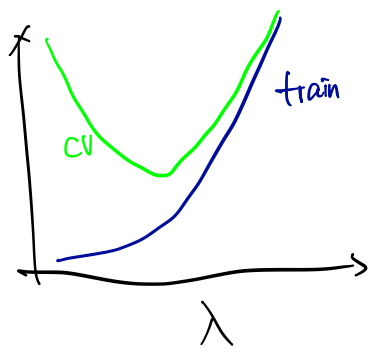
$J_{cv}(\theta) \gg J_{train}(\theta)$

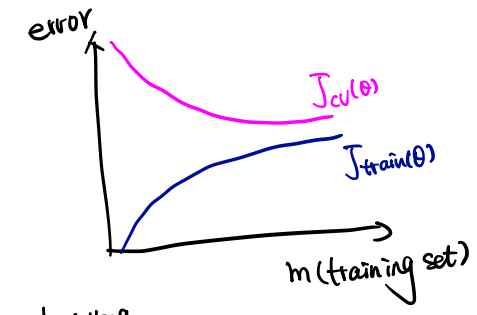
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{m} \theta_j^2$$

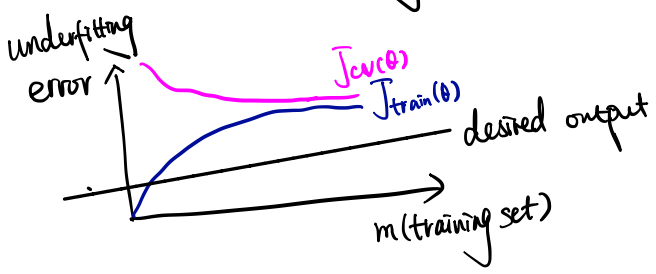
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x_{train}^{(i)}) - y_{train}^{(i)} \right)^2$$

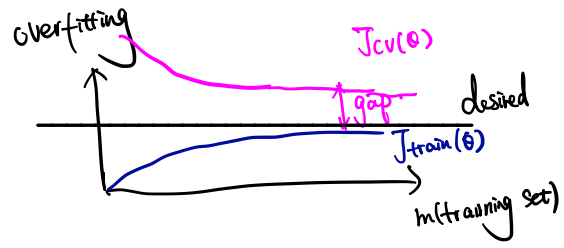
$$J_{cv}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2$$

$\lambda = \begin{cases} 0 \\ 0.01 \\ 0.02 \\ 0.04 \\ \vdots \\ 10 \end{cases}$









overfitting, increase trainning set is likely to help.

Get more traning examples : Fix high variance

try small sets of features : Fix high variance

try extra features : Fix high bias

adding polynomial features Fix high bias

Decreasing λ Fix high bias

Increasing λ Fix high variance

Building Spam classifer.

prioritizing what to work on:

- collect lots of data
- Develop sophisticated features (using email header data in spam emails)
- Develop algorithms to process your input in different ways (recognizing misspellings)

Recommend approach:

— start with simple algorithm you can implement quickly.
Implement it and test it on your cross validation data.

— Plot learning curves to decide if more data, more features. etc are likely to help.

— Error analysis: Manually examine examples (in cross validation set) that your algorithms made errors on. See if you spot any systematic trend in what type of examples it is making errors on).

skewed dataset problem:

Accuracy: correct cases / all cases

precision: $P = \frac{TP}{TP+FP}$

recall: $R = \frac{TP}{TP+FN}$

	Relevant	Non-relevant
Retrived	TP	FP
Not Retrived	FN	TN

Last Letter stands for Predicted
Combined Letter stands for actual

$F_1 = \frac{2}{\frac{1}{R}+\frac{1}{P}} = \frac{2PR}{P+R} = \frac{2TP}{2TP+FP+FN}$

$F_{measure} = (\beta^2+1)\frac{P+R}{\beta^2 P+R}$

Trading off precision and recall.

predict 1 if $h_\theta(x) \geq 0.9$

predict 0 otherwise

higher precision, lower recall.

vice versa.

precision- recall curve.

threshold affects precision & recall.

Large data rationale:

Useful test: Given input x, can human expert confidently predict y. If can, a large dataset can help.

1. use low bias algorithms.
2. use very large trainning set → low variance

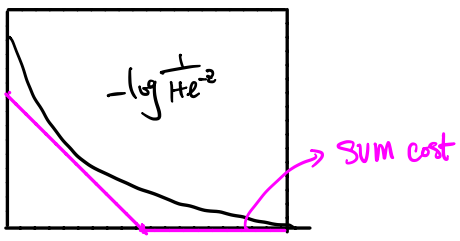
A large trainning set is unlikely to help when:

① humans cannot predict accurately.
② too large number of features (low bias)
③ we're not using regularization.

1. human test.
2. large low bias algorithm + large training set.
can ensure high accuracy.

SVM.

logistic regression cost:



logistic regression cost:

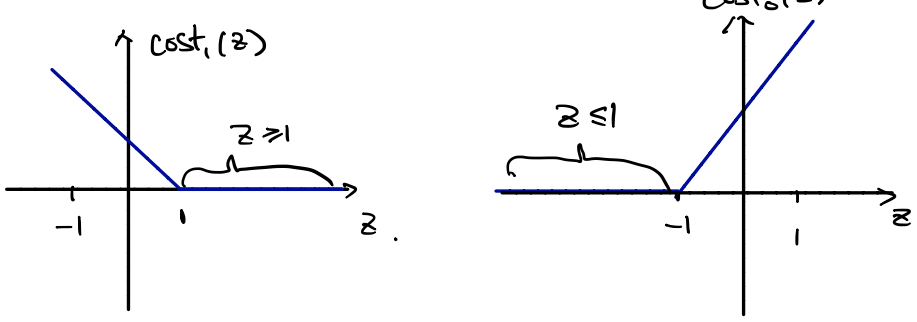
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \cdot \log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

SVM: (Large margin classifier)

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

↓ C is "$\frac{1}{\lambda}$" also controls regularization.

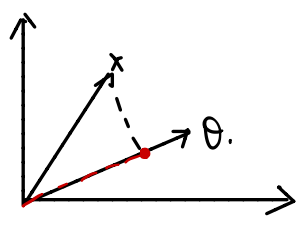
$h_\theta(x)$ { 1 if $\theta^T x \geq 0$ For SVMs.
0 otherwise.



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

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

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



Decision boundary is always perpendicular to θ.

Intuition:

$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\|\theta\|^2$ p is projection x onto θ.

s.t. $p^{(i)} \cdot \|\theta\| \geq 1$, if $y^{(i)}=1$
$p^{(i)} \cdot \|\theta\| \leq -1$, if $y^{(i)}=0$

Kernel for SVMs:

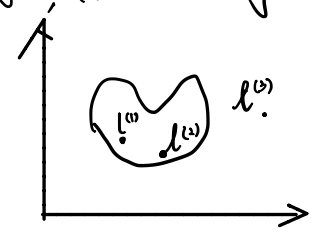
high polynomial feature function:

choose lemma $l^{(1)}, l^{(2)}, \ldots l^{(3)}$

Gaussian kernel:

$f_1 = \text{Similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x-l^{(1)}\|^2}{2\sigma^2}\right)$

bigger σ^2, the extend get more spread out.



(logistic regression does not work well with kernel)

choose lemma l as training set.

we get feature vector

$f_1^{(i)} = \text{Similarity}(x^{(i)}, l^{(1)})$ $f = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_m \end{bmatrix}$ $f_0=1$

predict "y=1" if $\theta^T f \geq 0$

Training:

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

$\sum_j \theta_j^2 = \theta^T\theta \Rightarrow \theta^T M\theta$ This is implemented to allow fast calculation.
(ignore θ_0)

$C(=\frac{1}{\lambda})$. Large C: low bias, high variance
small c: high bias, low variance.

σ^2. large σ^2: Feature vary more smoothly. higher bias, lower variance.
smaller σ^2: higher variance, low bias

Need to specify: ① Choice of parameter C
② choice of kernel.
Gaussian kernel.: need to choose σ^2

Note: perform feature scaling before using the Gaussian kernel.

(need to satisfy technical condition called "Mercer's Theorem" to converge.

Kernel available:

- Polynomial kernel. $k(x,l) = (x^T l + \text{const})^{\text{degree}}$
- More esoteric: String kernel, chi square kernel, histogram intesection kernal...

For multiclass, rest vs all.

n = # of features. m = # of training examples

if n is large. (compare to m) ⇒ logistic regression, linear kernel.

if n is small. m is intermediate. Use SVM with Gaussian Kernel.

if n is small, m is large. add on more features. use logistic R or linear kernel.

Unsupervised learning:

Clustering algorithm:

K-means clustering algorithm:

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

k clusters

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

K-means algorithm

Randomly initializes k cluster centroids $\mu_1, \dots, \mu_k \in \mathbb{R}^n$.

Repeat {

for $i = 1$ to m

$c^{(i)}$:= index (from 1 to k) of cluster centroid closest to $x^{(i)}$ $\left(\min_k \|x^{(i)} - \mu_k\|^2\right)$ → $c^{(i)}$

for $k = 1$ to k

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

} if one cluster have 0 points, then it should be eliminated or reinitialize it.

K-means algorithm objective.

$c^{(i)}$ = index of cluster which $x^{(i)}$ is assigned.

μ_k = cluster of centroid k ($\mu_k \in \mathbb{R}^n$)

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ assigned

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

if J increase, there is something wrong in algorithm.

K-means random initialization.

local optima.

using training example $x^{(i)}$ as μ_k.

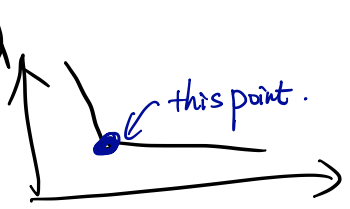
more random initialization. and choose one have least cost.

For $k \leq 10$, multiple random initialization may help a lot but for $k \geq 10$, it may not help so much.

Choose number of K.

usually manually picking.

Elbow method



or depend on downstream purpose / human insight later.

Data Compression.

Data compression.

PCA (Principal component analysis)

Find k vectors $u^{(1)}, u^{(2)} \dots u^{(k)}$ onto which to project data, so as to minimize error.

span of

Cost function: $J = \frac{1}{m}\sum_{i}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2$.

Principle Component Analysis Algorithm.

Reduce data from n dimensions to k-dimensions

Compute "Covariance matrix":

sigma $\longrightarrow \Sigma = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)})(x^{(i)})^T \Rightarrow (n \times n)$ matrix

Compute "eigenvectors" of matrix Σ:

```
[U, S, V] = svd(Σ);
```

$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & & | \end{bmatrix} \Rightarrow$ we get first k columns of data.

$\in \mathbb{R}^{n \times n}$.

$z = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix}^T x = \begin{bmatrix} — (u^{(1)})^T — \\ — (u^{(2)})^T — \\ \vdots \\ — (u^{(k)})^T — \end{bmatrix} x^{(i)} \Rightarrow (k \times 1)$ matrix.

note:

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

Octave Implementation:

```
Σ = (1/m) * X' * X;
[U, S, V] = svd(Σ);
Ureduce = U(:, 1:k);   ignoring x0 = 1.
z = Ureduce' * x;
```

Reconstruction from compressed data:

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

$(n \times 1) \quad (n \times k) \quad (k \times 1)$.

how to choose k:

Choose k to be smallest:

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

I would say 99% percent of variance is retained.

```
[U, S, V] = svd(Σ)
```

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

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

advice for applying PCA:

Supervised learning speedup.

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

↓

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

mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on training set.

Application of PCA:

- Compression:
  - Reduce memory
  - speed up learning algorithm.
- Visualization:

It is not a good way to avoid overfitting.

before implementing PCA, try running with raw data first.

if cannot, can use PCA.

Anomaly detection:

problem: x_{test} is anomalous?

$p(x_{test}) < \varepsilon \rightarrow$ flag anomaly

$p(x_{test}) \geq \varepsilon \rightarrow$ ok.

Fraud Detection.; Manufacturing ; Monitoring computer

$x^{(i)}$ = features of i's activities

Moder $p(x)$

Judge / Identify.

Guassian distribution:

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

Parameter estimation: $N(\mu,\sigma^2)$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu)^2$$

($\frac{1}{m-1}$ in statistics)

Anomaly detection algorithm,

1. Choose features that you think might be example of anomaly.

2. Fit $\mu_1 \cdots \mu_n, \sigma_1^2 \cdots \sigma_n^2$.

$$\mu_j = \frac{1}{m}\sum_{i=0}^{m} x_j^{(i)} \qquad \sigma_j^2 = \frac{1}{m}\sum_{i=0}^{m}(x_j^{(i)}-\mu_j)^2$$

3. Given new example, Compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j;\mu_j,\sigma_j^2) = \prod_{j=1}^{n}\frac{1}{\sqrt{2\pi}\sigma_j}\exp\left\{\frac{-(x_j-\mu_j)^2}{2\sigma_j^2}\right\}$$

Nomaly evaluation

tranning data: unlabelled 6000 good engines (can contain some anomaly)

CV. labelled dataset 2000 good 20 bad.

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (ok)} \end{cases}$$

possible evaluation metrics: Precision, recall, F_1-score.

choose ε on valid set.

Anomaly detection vs. Supervised learning:

Anomaly detection	supervised learning
① very few anomaly ② large number normal ③ many types of anomalies, future ones might be never met before learn from normal ones	① large number of positive & negative ② enough nomally to learn what nomally looks like.

What data to use.

① make feature more guassian, plot it in hist.

$\log(x)$. $x^{0.5}$

② ① more feature distinguish anomaly from normal.

② create new features from existing features.

to address this problem.

use Multivariate Guassian (Normal) distribution

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

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

$$\Sigma = \begin{bmatrix} \sigma_1^2 & & 0 \\ & \sigma_2^2 & \\ 0 & & \ddots\ \sigma_n^2 \end{bmatrix}$$ is original model.

original model: assume no correlation between different models.

Computationally cheap.

For multivariable guassian model: must $m > n$, or Σ is non-invertible

if Σ is noninvertible:

① $m < n$

② linearly dependent features.

Recommendation systems:

Movie recommendation:

n_u = no. users n_m = no. movies $r(i,j)=1$ if user j has rated movie i.

$y^{(i,j)}$ = rating given by user j to movie i.

Problem formulation:

$r(i,j)=1$ if user j has rated movie i (0 otherwise)

$y^{(i,j)}$ = rating by user j on movie i (if defined)

$\theta^{(j)}$ = parameter vector for user j

$x^{(i)}$ = parameter vector for movie i

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

$m^{(j)}$ = no. of movies rated by user j

to learn $\theta^{(j)}$:

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

Total optimization learning algorithm:

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

(n parameters)

all the movie to one user

Gradient descent update:

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

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

pretty much as logistic regression.

Collaborative Filtering:

if user $\theta^{(1)} \dots \theta^{(n_u)}$, to learn $x^{(i)}$:

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

to learn $x^{(1)} \dots x^{(n_m)}$:

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

↓ This is j. now.

Guess $\theta \to x \to \theta \to x \to \theta \dots$ simultaneously learn θ and x.

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

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

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

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

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

Note: There is no $x_0=1$ here. so we do not write like that.

3. predict. star rating $\theta^T x$.

$$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \vdots \\ -(x^{(n_m)})^T- \end{bmatrix} \quad \Theta = \begin{bmatrix} -(\theta^{(1)})^T- \\ \vdots \\ -(\theta^{(n_u)})^T- \end{bmatrix}$$

predicted ratings: $X\Theta^T$ Low rank matrix factorization.

Find related movies:

small $\|x^{(i)} - x^{(j)}\| \Rightarrow$ similar movies.

mean normalization:

minus the mean value in that column,

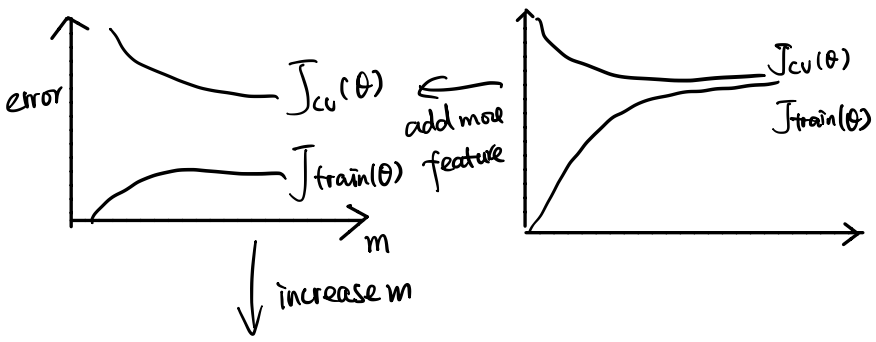
For user j, the rating score,

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

because ratings are already comparable, so do not need to do range normalization.

Machine learning with large dataset.

with small dataset first.



work!

Stochastic gradient descent: (use 1 example each iteration)

(Batch gradient descent. whole batch.) whole dataset.

1. random shuffle (reorder) training examples
2. repeat { (for some iterations)
   for $i := 1, \cdots m$ {

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

(for all j)

   }
}

Mini batch gradient descent:

let $b=10$, $m=1000$.

Repeat{

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

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

}
}

one advantage parallel computation.

checking for convergence.

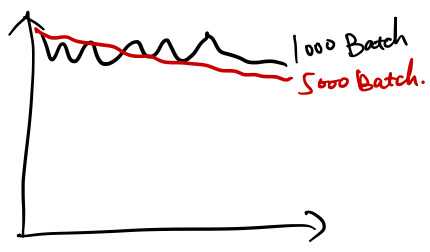
SGD:

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

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

Every 1000 iterations, plot average of cost.

checking for convergence: for $J(\theta)$.



Online learning:

Product search

User search for "Android phone 1080 camera"

give 10 results. 10 training pair.

learn $p(y=1|x;\theta)$. update parameters.

① change update with time. of user need.

② update 1 at a time. and disgard that training set.

Map-reduce: (Jeffery Dean) legendary person google.

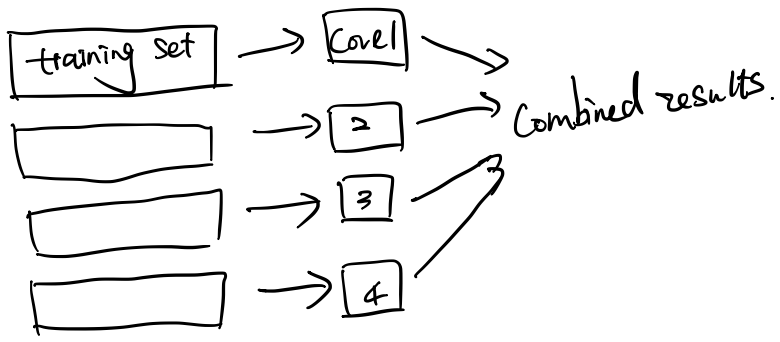
Split work evenly to training set.

different machines and send to one central machine.

Map-reduce needs learning algorithm to perform summation over training set.

Many learning algorithm can be expressed as computing sum of functions over training set.

Multi-core machine:

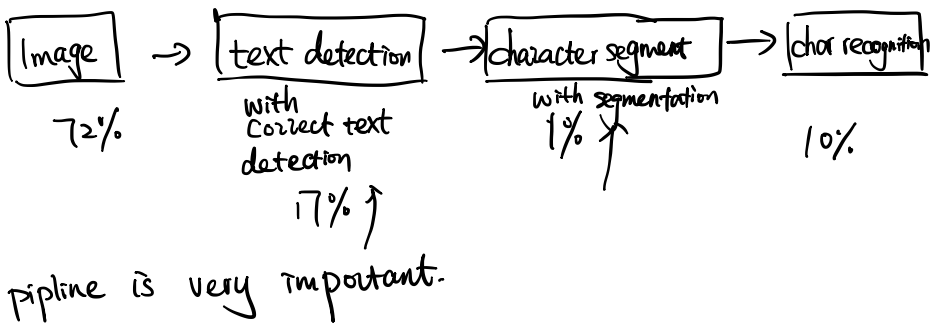


OCR
  pipline
Sliding windows

Artificial data:
  ① Create Data (OCR font)
  ② Data amplification. (Audio, background knowledge)

Data:
1. make sure you have a low bias classifier before expanding effort. (plot learning curves)
E.g. keep increasing # of features/hidden units in neural network until you have a low bias classifer.
2. how much work would it be to get 10x as much data as we currently have.
  — Artificial data synthesis
  — label/collect yourself.
  — "crowd source" (E.g. Amazon Mechanical Turk)

Ceiling analysis:
  give correct labels:



pipline is very important.

Summary: Main topics

Supervised learning:
  Linear regression, logistic regression, neural networks, SVMs.

Unsupervised learning:
  K-means, PCA, Anomaly detection.

Special Applications/special topics
  — Recommender system, large scale machine learning.

Advice on building a machine learning system
  — Bias/Variance, regularization, deciding what to do next.
  evaluation of learning algorithms. learning curves
  error analysis. ceiling analysis.