W13 - Support Vector Machines

13_93. Optimization objective __Support Vector Machines

within Machine learing, supervised algorithm performance will be similar, and what matters **more will be things like the amount of data, and skills in applyng this algorithm: like features choosen, and regularization paramter.**

SVM (support vector machine): is a powerful supervised algorithm.
Compared with linear regression / logistic regression, gives more powerful way of learning **complex nonlinear functions.**

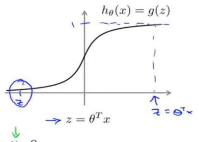
1. Alternative view of logistic regression

1.Logistic function:

if y=1, want h(x)=1-------> θ^T*x >> 0
if y=0, want h(x)=0-------> θT*x << 0

Alternative view of logistic regression

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



If $y=1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$
If $y=0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

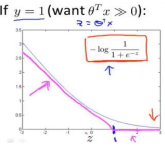
2. Logistic regression cost:

Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

each signal training example, contrib $= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y)\log(1-\frac{1}{1+e^{-\theta^T x}})$

>2.1 when y=1 , cost function is: J=log(1+e^-Z) (=-log (h(x))):
When y=1, to min J, try to set Z=θT*x >> 0

If $y=1$ (want $\theta^T x \gg 0$):



Cost_1(z): Modify cost fuction to the combination of two straint line-purple li
>> Pretty close to logistic regression cost function;
>> Flattet line + straining line
>> have similar function as logistic cost funciton,
but give SVM computional advanctage + easiser optimzation problem.

>2.2 when y=0 , cost function is: J=log(1+e^-Z) + Z (=-log (h(x)) +Z):
When y=0, to min J, try to set Z=θT*x << 0

Cost_0(Z): Modified cost function as above

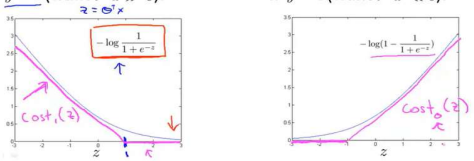
(note:
1. computional advantage: loss function now is much simplier, easier to compute
2. less strict for minimizing cost loss: logistic require z >>0 or <<0, while now only require z >1 or <-1, much easier to get this threshold |1|;
3. for 2 class classification, if still use sigmoid activagiton function, then only required probability: output y^>1/(1+e^-1)=0.73 , will think it as 1._previous/entry corss loss will force probability as large as possible.)

Alternative view of logistic regression

Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

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

If $y=1$ (want $\theta^T x \gg 0$): If $y=0$ (want $\theta^T x \ll 0$):



3. Support vector machine

cost function: modfied on logistic cost function: remove 1/m, anoterher type of weith on paramter:

sum y(i)* cost_1(z(i)) + (1-y(i)) * cost_0 (z(i)) over all training example + λ * sum of all paramter^2 *regularizaitn term)
=A+ λ* B
=C* A +B (while C play a role of 1/λ)

just a different way of trading off, parametrizing how much care about optimizing the first term A vs second term B.
C play a role similar of 1/λ, while not equal to 1/λ (otherwise, optimized paramters θ will be same under A+ λ B and J= C* A +B).

Object function_SVM: min C*A+B

Support vector machine

Logistic regression:

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

A B

Support vector machine:

$$\min_\theta C\sum_{i=1}^{n} y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)}) + \frac{1}{2}\sum_{i=0}^{n}\theta_j^2$$

$\min_u (u-5)^2+1 \Rightarrow u=5$ | $A+\lambda B$ | $C = \frac{1}{\lambda}$
$\min_u 10(u-5)^2+10 \Rightarrow u=5$ | $CA+B$

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

4. SVM hypothesis:

Hypothesis: does not output the probability. With optimized paramter θ by object function, will just make a prediciton: y=1 or 0.

h(x)=1, if θT*x >= 0
h(x)=0, otherwise
(note: cost1 (z) = k*z -k; cost = -k*z-k-->
dz = y*k - (1-y)*k = k (if y =1);
dz = -k (if y=0)
-->parameter updating step /penalt for differance between output and labeled is constance
if y=1, z<0-->y^=0,-->dz = k
if y=1, 0<z<1-->y^=0-->dz =k, even now y^==y: still updating parameter till dz=0<--z>=1
updated parameter make z have safety maregin: 0<z<1, in this region output is correct, while cost function still force algorithm to update till z>=1
)

SVM hypothesis

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

Hypothesis:

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

Note:
1. decision boundary comparision: logistiv vs SVM
logistic regression: -->decision boundary: W*X +b = a(decided by probability threshod, to judge as 1)
SVM: decision boundary: Ws*X +b_s = 0
margin boundary: Ws*X +b_s = 1
seems SVM have a 'safety ' margin, but not more complex decions boundary..

2. SVM : is actually 'linear function, prediciton'
but cost funciton: not using squared error (could not penalt prediction differance-not convex) or entry-cross or softmax, using a new stype cost function instead.
2,1: if cost function using entry_cross or squared error, then loss if only f(y), could not transfer to dz
as y=1, z>=0
so use 'special' cost function f(z)

13_94. Large Margin Intuition __Support Vector Machines

SVM also called large margin classfiers.

1. SVM:

Min J c*A+B-->
if y=1, J=0-------> θT*x >= 1
if y=0,J=0-------> θT*x <= -1

2. SVM property:
When y=1, if θT*x>=0, then h(x) will predict 1,
when y=0, if θT*x < 0, then h(x) will predict 0.

SVM wants more, don't just barely get the example right. but also want cost fucnton =0., then require big/less than 0:
When y=1, if θT*x>=1,
when y=0, if θT*x < -1,

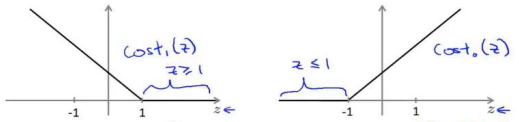
--->builds in an extra safety / margin factor into SVM

Note: below conclusion(withoug regulization) , right or not?
1. decision boundary (right prediction y^): θT*x=0
2. safety margin(cost error):
>1. positve examples: θT*x>=1;
>2. Negative examples: θT*x<=-1;
Gap between θT*x=0 and θT*x=1 is the margin between decision boundary and positive examples;
Gap between θT*x=0 and θT*x=-1 is the margin between decision boundary and negative examples;

Note:
this kind of cost functtion, when prediction is far away from labeled y, will push θT*x<=-1 or θT*x>=1 (gradient step dz = k), have a margin from decison boundary, θT*x=0; **why need this margin??**

Support Vector Machine

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



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)

C = 100,000

3. SVM Decision Boundary: 'C' affect on margin factor:

SVM object : min C *A +B = C* sum of all examples (cos_1 +cost_0) +B

When C is larger make A=-0: θT*x >= 1 when y=1, θT*x <= -1, when y=0.

if think of optimaztion problem now is choosing parameter θ to make sure A=0, then object now is constrained to :

Decison boundary:
min B (sum of parameters' square θ_i^2)
strict: θT*X >= 1 if (i) y=1;
θT*X <= -1 if (i) y=0.

4. SVM Decision Bounday: Linearly separable case

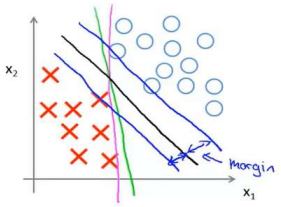
Linearly separable: exist a straight line/many a different straight lines, can seperate the positive and negative examples perfectly.

When C is large:
SVM decison bounday:
SVW will a decsion boundary-black line in right pic:
> - Seems more robust seperate the positive and negative examples;
> - Larger minimum distance from any of the training examples. (whearase the pink/green lines more close to training examples-did less good job seperating positive and negative classes.)

SVM Margin: the minimum distane of SVM boundary to any of the training examples.
-->gives SVM certain robust because it tries to separate the data with as large a margin as possible.
(note: optimized parameter θ, meet θT*X>=1 /<=-1, while decision boundary is θT*X=0
e.g: X=(1, x1, x2), θ=(θ0, θ1, θ2)
-->decison boundary l0: : x2 = -(θ1/θ2)*x1-((θ0/θ2)
-->stric line l1: : x2 = -(θ1/θ2)*x1-((θ0/θ2) + 1
-->strict line l_-1: : x2 = -(θ1/θ2)*x1-((θ0/θ2) -1
when C is very large, required A ==0-->θT*X>=1 /<=-1-->SVM margin >=1
)

-->also called large margin classifer.
<-----Consiquence of above optimization problem (when C is very large).

SVM Decision Boundary: Linearly separable case



Large margin classifier

5. Large margin classifier in presence of outliers

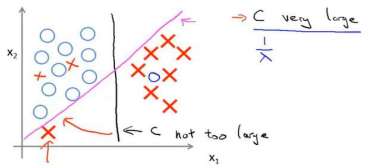
worke out a large margin classifer in the case of only when C is very large,right pic. Black line.
SVM is actually slightly more sophisticated than right pic large margin view might suggest.

If use large margin classifer, then learnig algorithm can be sensitive to outliers (异常点).
(as require strictly A=0--> for all training examples: θT*X >= 1 if (i) y=1; θT*X <= -1 if (i) y=0.
-->θ will be changed by outliers,--> decision boundary θT*X =0 changed.)
e.g . in right pic, when added one outlier, the boundary may change from black line to pink line to get largest margin to training examples.
And it is realy unclear, based on a single example/outlier, it is actually a good idea to change decision boundary from black one over to the pink one.

when C is reasonable small, then will end up still the black decision boundary.
(C is reasonable small, need balance CA and B to get min C*A +B, do not require A =0, could accept A>0 for outliers)

if data is not linearly separable (have positve among negative, and negative among positive), SVM will also do the right seperating.

Large margin classifier in presence of outliers



Margin classifier is only when C is very large, play a role similar to 1/λ(regularization parameter); but when C is not so large, SVM can ignoring the few outliers , and also do fine even if the data is not linearly separable.

13_95. The mathematics behind large margin classification __Support Vector Machines

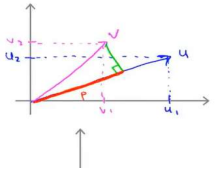
1. Vector Inner Product:

Vector: v, u.
Vector inner product: $v^T * u = u^T * v$

vT* u = u T * v = p_v* ||u|| = p_u * ||v|| = v1 x u1 + v2 x u2.

||u|:length of vector u= sqrt (u1^2 + u2^2)
p_v: length of vector v projection on vector u. could be positive and negative.
p >0 when angle betwen two vector less than 90 degree. P<0 angle greater than 90 degree.

Vector Inner Product



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

$u^T v = ?$ $\begin{bmatrix} u_1 & u_2 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

$\|u\|$ = length of vector u $= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$

p = length of projection of v onto u.

$u^T v = p \cdot \|u\| = v^T u$ $p \in \mathbb{R}$

Signed

2. Decison boundary: when C is large:

min B (sum of parameters' square θ_i^2)
<2.1 strict: θT*X >= 1 if (i) y=1;
θT*X <= -1 if (i) y=0.

simplication: let **θ_0 =0 , n =2 (input feature number)**

>2.2. Objects: min B =min 1/2* (θ_1^2 + θ_2^2) ~ same: min **=1/2 * ||θ||^2.**

SVM object is mimizing half of square of paramter θ norm

>2.3. Stricts:

strict: θT*X >= 1 if (i) y=1;
θT*X <= -1 if (i) y=0.

for one single psositive example x(i), y(i)=1, θT* X(i)

θT* X(i) = P_x(i) * ||θ|| >=1
P_x(i): projection of x(i) onto vector θ

while make object small under strics---**>||θ|| small and P_x(i) * ||θ|| >=1 / <=-1**

paramter vector θ: is the vertical vector of decision boundary.
(vector θ=(θ1, θ2)-->
line: θT*(1, x1, x2) =0/1/-1 --->
x2 = -θ1/θ2 * x1 +1/1/-1: -->
line direction vector: (1, - θ1/θ2)-->a=(θ2, -θ1)
-->a*θ =0: line directoin vecotor and parameter vector θ is vertical to each other)

-->P_x(i)=||x|| * cost (angle between vector θ and vector x(i)) should be large
(利用向量求空间中点到直线的距离是有公式的:
定理: 给定R^3] 中直线 l , 其方向向量为 n . A为 l 外一点, 若要求 A 到直线l [公式] 的距离 d , 可任取 l 上一点 B , 点 A 到点B的向量记作m , 则
d = |n.m|/ ||n||)??--n 应该为直线法向向量。。。)

(training set x(i) distance to decison boundary θT*x =0:
take B =(0,0)_ θ0=0-->
d = |x(i).θ|/||θ|| = P_x(i)/||θ||
)

strict: θT*x = P_x(i) * ||θ|| = d* ||θ|| >=1
object: ||θ||^2
-->d need to be large:
decision boundary line should be far away from training data:

d= P_x(i)/||θ|| = |x(i)|* cos (a) /||θ||: |x(i)| is fixed , ||θ|| small, cos(a) should be large
--->& angle between vector θ and x(i) small to arond 0 (y=1) or large to around 180 (y=0) ---->decison boundary line should close to vertical to training data

3.1 e.g: first pic. In right: choose green line decision boundary:

P_x(i) = ||x|| * cost (a) is small value,

due to cost (a) is too small--->θ direction /decision boundary direction need to change.

3.2 e.g: 2nd pic. In right: choose green line decision boundary:

P_x(i) = ||x|| * cost (a) is big value,

due to cost (a)~ =1 for postive x(i), cost(a)~c=-1 for negtive x(i).
θ direction now is horizental, angle of examples' vector and θ direction is close to 0 (positve example) /180 (negative example).

By choosing decsion on the 2nd pic, exmaple x(i) projection length on parameter vector θ , ||x|| * | cost (a)| can be much bigger-distance to decision boundary is bigger , **SVM can make paramter norm ||θ|| much smaller .--> SVM object 1/2 * ||θ|^2 is smaller.:**
object: d*/||θ|| (make d bigger, while ||θ|| small)

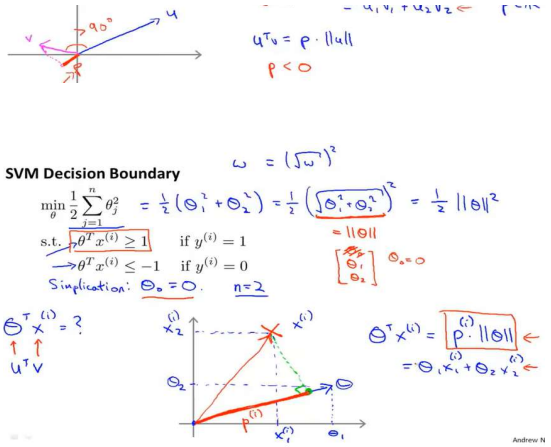
(note: SVM:
Make right prediction θT*x >=0 y=1,
while make the decision bounday far from training set: by cost function defined 0: when d* ||θ ||>=1)

Above is how SVM gives this large margin classification effect.
If want project p_x(i) of positive and negative on vector θ to be large, the only way to hold true is there is large gap that seperates postive and negative examples.
And the maginitude of this margin is exactly the values of P_x(1), P_x(2), P_x(3) and so on.

by making this value C large (no regulization?), SVM end up with a smaller value of norm of θ, which is what it is trying to do in the objective (smaller norm of θ-->incase overfitting).
also why SVM ends up with enlarge margin classifiers: try to large the norm of these P_x(i) which is the distance from the training example to the decsion boundary.

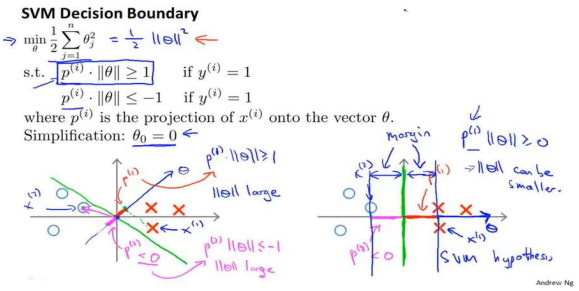
Note:
when C is larege, and θ_0 /=0,
and when still have above object and strics, (rigth objects and strics)

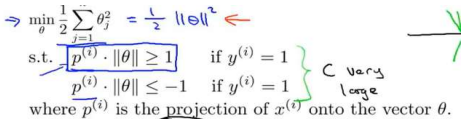
SVM still try to find large margin separator between the positve and negative



decision boundary: θT*X=0:
vertical direction: θ: parallel with most training data vector;
distance to training data: large distance to negative and positive examples.

Note: ??
1. if C is not very large, -->Object is minize C*A +B, : balancing minimizing A (θT*X, θT*θ), may not require θT*X>=1 / <=-1, distance and angle to decision boundary may not required so strictly-->more robust to outlets（异常点）
2. margin required: cost=0 while θT*X>=1 / <=-1. if define cost =0 while θT*X>=0 / <=0, then minimize object is : C*A +B. When C is very large , only required θT*X>=0 / <=0, distance and angel to decision boundary do not required so strictly-->could not get the largerst margin classfier, and more robust to outlets（异常点)?





13_96.Kernels I__Support Vector Machines

Adapting SVM in order to develpoing complex nonlinear classifiers.
Main tech. : Kernels

1. Non-linear Decsion Boundary:

for developong complex nonlinear classifers, method:

>Method 1. complex polynomial featurs:

Predict y=1 if :
θ_0 + θ_1*x1 + θ_2 * x2.....+θ_5 * x2^2 >=0

Problem:
High order polynomia is one way to come up with more features. but no idea if this high order polynomial is what we want.
and when comes to computer vesion, where input is image with lots of pixels, using high order polynomial is computionally expensive.

> Method 2. define new feature f1, f2..
Better choice of the features f1, f2, f3.

e.g: x_0 feature do not consider here.
Given landmarks l1, l2,l3--->

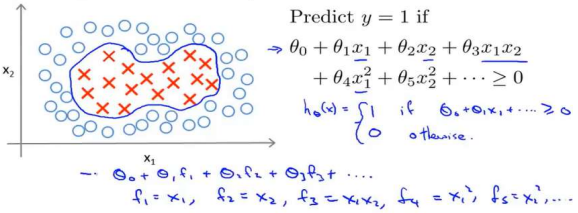
training example x(i) , feature will be relation/factor of distance to each landmarks:

f1_x(i) = similarity (x(i), l1) = exp (- ||x(i)- l1||^2/ 2/ σ^2) (range: 0-1)

- ||x(i)- l1||^2: euclidean distance of x(i) vector to landmark l1 vector.

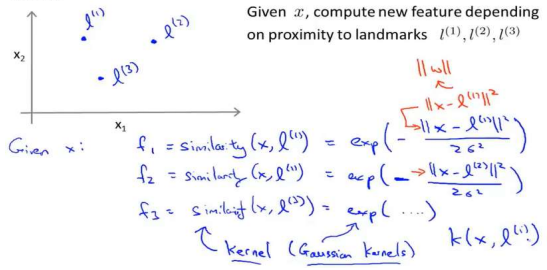
Kernal function k(x, l(i)): this similiarity function above.
and here Gaussian （高斯） kernal is used. also have other kernal functions except Gaussian.

Non-linear Decision Boundary



Is there a different / better choice of the features $f_1, f_2, f_3, \ldots$?

Kernel



2. Kernels and similarity

Note: ||x(i)- l1||^2 = x vector - l1 vector on all element/units (x (i)_0 not considered here)

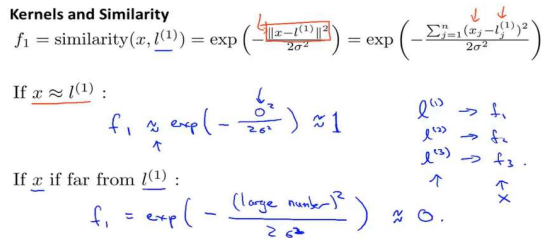
if x(i)~= l1 ---------> f1 ~=1

if x (i) is far from l2---------> f1 ~=0

feature f1, f2..:function:
> 1. measure how similar X(i) is from one of the landmarks:
f close to 1 when x close to landmark;
f close to 0 when X is far from landmark.

>2. Each landmark defines a new feature: similarity/distance to each landmark

Kernels and Similarity

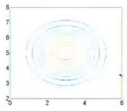


3. Kernal function intuiation:

> 3.1 Quantifiable plot (等高线图） :
make f1= a (constance)-->||x(i)- l1||^2=a'*σ^2

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

for a constant f1, x(i) is a circle (circle point is l1 point) and radius is a'* σ



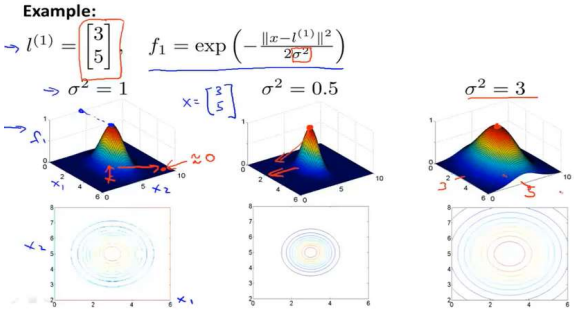
>3.2. Z^2=||x(i)- l1||^2 take different value-->f1=exp (-(z/σ)^2) （抛物

f1 =1 when x(i)=l1, and close to 0 when x(i) far from l1.

>f1 Measures how close x(i) to l1 landmark;
>σ: when σ becomes bigger
--> f1 decrease slower with Z^2 increase.
(-less sensitive to distance,, different examples' feature more close to each other, harder to seperate positive and negative, tend to underfitting.)
-->Quantifiable plot becomes larger (radius bigger, need bigger x/distance to get same f1)

>σ: when σ becomes smaller
--> f1 decrease faster with Z^2 increase.
(more sensitive to x/distance change, different examples' feature more seperate from each other, easier to seperate positive and negative, tend to overfitting)
-->Quantifiable plot becomes shrink / smallerr (radius smaller, need smaller x to get same f1/relationship factor)

Example:



4. Example using similarity features

Hypothes h(x) has trained paramter θ vector=[-0.5; 1; 1; 0]T, landmark l1, l2, l3 already know

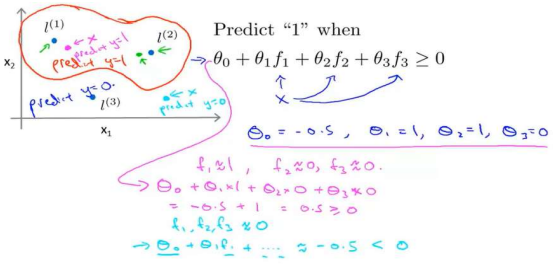
Function: prediton h(x) =1, when θ^T* [f1; f2; f3]T>=0 ;

computation: give right pic. pink example x, prediction output:

as x is close to l1, --> f1 ~=1, f2 = f3 = 0,
----> θT* [f1; f2; f3]T =0.5 > 0----->h(x)=1

Decision boundary: righ pic red curve is the boundary :θT* [l1; l2; l3]T>=k (prediton h(x) =1)
how to define threshhod k? k=0
(Note: decision boundary: θT*f : is measuring the sum of one sample(x1, ..xn)'s ditance to each training example)

Prediction boundary is decided by paramter and landmark (θT* f): depending on weights add to distance from each landmark)



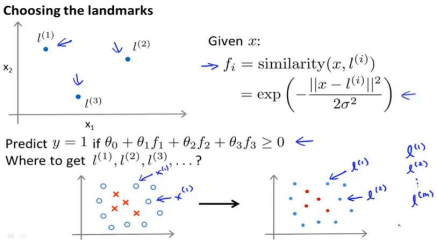
Kernel use in/ for supporting SVM:
Define extra features using landmakr and similarity function to learn more complex nonlinear classifiers.
(Decision boundary:
1. θT* x=d*|θ|>=0 : -->distance far from training set to decision boundary >0: cos(a): 0-1: a: -90 ~ 90 degree for y =1 (只要求postive在decision boundary 一侧， negative 在另一侧； cost function 要求距离d*|θ|>1)
2. kernal: θT* f>=0: -->sum of weighted distance of sample to each training set (positive : distance >0； cost function 要求'权重距离之和'>1)

13_97.Kernels II__Support Vector Machines

1. Choosing the landmarks:

Put landmarks: in practice exactly the same locations as the training examples.
Landmarks: l1, l2....lm (same number as training examples),
with one landmark per position of each training examples.

This is nice as features basically going to measure how close an example is to one of the things saw in training set.



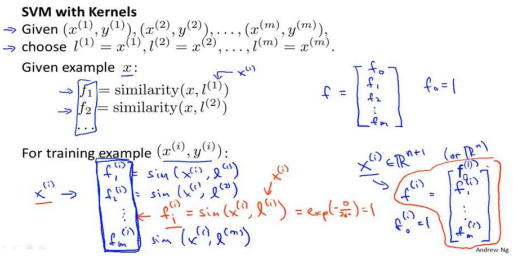
2. SVM with Kernel

training set: x(1), y(1)..... X(m), y(m):
choose l(i)= x(i) , i=1...m

2.1 Given example x (from training set or dev set or test set)-->feature vector

> compute X feature vector f _x : f_x_i= similarity (x, li)
f _x =[f0; f1; ...fm]^T, **f0=1**. 0<= f_x_i <=1

2.2 representing training example by feature vector:
example x(i), --> f(i)= [f(i)_0; ....f(i)_i; ...f(i)_m]T
note: f(i)_i = 1, f(i)_0 = 1

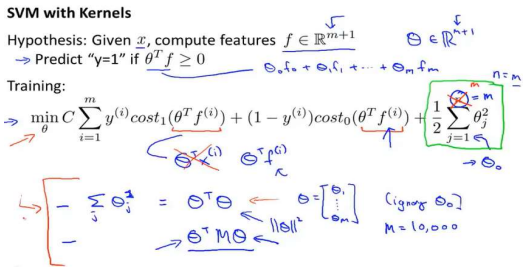


2.3: Predit on example X:

Process:
> Compute x feature vector (m+1 dimensional vector f(i)_0=1);
> Compute Θ^T* f_x (using already trained Θ)

2.4: Train parameter Θ

for SVM origianl objective function:
>1. replace x(i) vector by feature vector f_x(i)
> dimension number n=m (x(i) feature number now equal to f_x(i) number =training example numbrer
> In regularization, still do not consider θ_0(f_x(i)_0 -1).



2.5 regulization in SVM object:

regulzation term: =1/2 * ||θ||^2 = 1/2 θ^T* θ

sometimes repleas θ^T* θ by : θ^T*M* θ
M: matrix depending on kernael used.
--->this modified regulization term gives a sligthly different distance metric. Use a slightly different measure instead of minimizing exactly the norm of θ.
minimize something slightly similar to it (original regulization term), like a rescale version of the parameter vector θ that depends on the kernel.

advantage:
> 1. Above mathmatical detail, allow algorithm to run much more efficiently;
>2. Allows SVM to scale to much bigger training sets
when training set is very large, orginal object (regulization not modified: dimension: mx1) will be computionaly expensive.
Modified regulation term slighted changed objective, make it faster.

2. 6: Kernel application:

Kernel could not use in logistic regrssion :
computational tricks that apply for SVM don't generalize well to other algorithms like logistic regression.
Kernel on logistic regression will be very slow:
computational tricks like modified regularization term in objective and details of how the SVM software is implemented, SVM and kernel tend go particularly well together.
while could run kernel on logistical regression, but very slow, and will not take advantage of advanced optimization tehniques that peope have figured out for the particular case of running a SVM with a kernel.
????
(Kernel:-->feature used to train algorithm , is distance to training set.
SVM: object: /decison boundary is distance to boundary, and cost function: is also maximing distance-->may be that's why SVM + Kernel work well??
logistic: feature is defined by human...../idea, ..run on dev set-->choose feature. distance to training set should also ok to seperate positive and negative??)
)

use off the shelf software packages that people have developed, which already embody these numerical optimiazation tricks.

2.7 SVM parameters:

loss = C*A + B

> 1. C(1/λ) :
Large C: Lower bias, high variance. more Prone to overfitting
(require strictly θT*x>=1 (for labeled y=1), θT*x<=-1 (for labeled y=0), -->tend to overfitting training set;
C increase, A need to decrease , make θT*x increase, make θ increase & angle change -->||θ|| bigger, algorithm tend to overfitting)
low λ-reduce regulation effect, helps for underfittiong, worse for overfitting

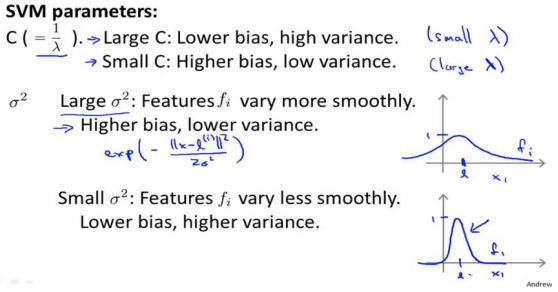
Small C: high bias, low variance. more prone to underfitting
high λ- increase regulation effect, worse for underfittiong, helps for overfitting
(require less strictly θT*x>=1 (for labeled y=1), θT*x<=-1 (for labeled y=0), -->tend to more robust to outlier, prone to underfitting;
C decrease, A could be bigger-->θT*x could smaller <1, make θ small or angle change-->||θ|| smaller algorithm tend to underfitting)
)

>2. σ^2: ???
Large σ^2: feature fi vary more smoothly , higher bias, lower variance.
feature f(i) less sensitive to training example distance x(i) with landmark, could not represented training example x(i) distance well and make the distance smaller-->algorithm trained on these feature f(i), hard to seperate positive and negative training set x(i)-->under fitting.

Gaussin curve tend to fall of relatively slowly, smoother function that vary more smoothly. give hypothesis high bias and low variance. as hypotheis will change slowly when input x changes.

Examples dirstribution areas in new feature space is smaller, harder to separely compleltey positive and negative.,(prone underfitting)

less sensitive to distance change/increase-example x position (could not capture example x feature sufficently), tend to underfitting



Small σ^2: feature fi vary less smoothly , lower bias, higher variance.
exmaples with new features locate in larger area in the new feature space---> get large margin between new examples, easy seperate positive and negative, tend to overfitting.

underfitting: algorithm could not fit training set well
overfitting: algorithm fit training set very well, yet could not fit dev set well (learned fake features of training set).

Note:
similarity function do not decided SVM decision boundary.???
SVM decision boundary: θT*x =0 / θT*f =0

13_98. Using an SVM___Support Vector Machines

1. **Use open source** instead of writing code from scrash.

Use SVM software package (e.g. libilnear, libswm, ...) to solve for parameters **θ;**

Need to specify:
> Choice of parameter C;

> Choice of kernel

>1. No Kernel ('linear kernel'): do not use kernel.--->standard linear classfier
predict 'y=1, if θ^T*x >= 0'

usage condition:
linear kernel used when training example have small size m, but large feature n for each example.--->SVM then is linear classfier, and not try to fit a very complicated nonlinear function.
decision boundary now is : θ_0 + θ_1 * x1 + θ_2*x2+...θ_n * xn >=0

(note: algorithm parameter/feature vector dimension is n+1: big enough to fit data set)

Use SVM software package (e.g. liblinear, libsvm, ...) to solve for parameters θ.

Need to specify:
→ Choice of parameter C.
Choice of kernel (similarity function):

E.g. No kernel ("linear kernel")
Predict "y = 1" if $\theta^T x \geq 0$ → n large, m small $x \in \mathbb{R}^{n+1}$

Gaussian kernel:
$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$.
Need to choose σ^2.

$x \in \mathbb{R}^n$, n small and/or m large

>2. Gaussian Kernel
l_i = x(i);

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

need to choose 6^2: influece bias and variance

usage conditon: when example feature number N is small, and m is large.
(note: so algorithm parameter/feature vector dimension is m+1: big enough to fit data set)

note: compare n vs m: then decide use linear kernel or Gaussian Kernel??

---

2. Kernel functions:

when use open source, normally asked to provide the kernel function.
If decided to use Gaussian kernel:

> write below code to compute f_i value (signal value):
-->generate all feature vectors and train SVM software.

x1: training or dev or test example
x2: landmark: **from training example**

```
function f = kernel(x1,x2)
    f = exp( -|| x1 - x2 ||^2 / (2σ^2) )
return
```

Kernel (similarity) functions:

```
function f = kernel(x1,x2)
    f = exp( -|| x1 - x2 ||^2 / (2σ^2) )
return
```

→ Note: Do perform feature scaling before using the Gaussian kernel.

Note: do perform feature scaling befor using the Gaussian kernel.

f_x(i) = [f_0; f_1; ...f_m]T

f_j =|| x(i)-l_j ||^2 = (x(i)_1 - l_j_1)^2 + (x(i)_2 - l_j_2)^2 + ...(x(i)_n - l_j_n)^2

if x(i) have big different values in different feature, x(i)_1=10,0000, x(i)_2=1.5 and close to l_j_2,
then x(i)_2 feature influence will much smaller/presented in feature element of f(i)_j, which will then be dominated by feature x(i)_1.

Do feature scaling, will make sure SVM gives comparable amount of attention to all of different features.

in this way,1. performe feature scaling first on all training set x(i)-->x'(i) , 2. then set them as landarmark l(i),3. then transfer training set x'(i) to similarity feature f(i)

---

3. Other choices of kernel

Not all similarity functions make valid kernels.
(Need to satisfy technical condition called 'Mercer's Theorem to make sure 'SVM packages' optimizations run correctly, and do not diverge).

This is because SVM implementation have many clever numerical optimization tricks, in order to solve for the parameter θ efficiently. In the original design envisaged, those decision made to restrict our attention only to kernels that satisfy this thechnical condition called Mercer's Theeorem, which make sure all the SVM packages can use large class of optimizations and get the parameter θ very quickly.
end up people ofen use linear kernal or Gaussin kernal. there are a few other kernels that also satisfy Mercer's theorem, but use less ofently.

Other choices of kernel

Note: Not all similarity functions $\text{similarity}(x, l)$ make valid kernels.
→ (Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

Many off-the-shelf kernels available:
- Polynomial kernel: $k(x, l) = (x^T l)^2$, $(x^T l)^3$, $(x^T l + 1)^2$, $(x^T l + 5)^6$, $(x^T l + \text{constant})^{\text{degree}}$
- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ... $sim(x, l)$

Many off-the-shelf kernel (similarity function) available:

> Polynomial kernel; lost of type.
could use inner produce, or cubic..(X^T* l + constant) ^degree

have two parameters: 'constant' number to add with inner product and power degree

Usage: usually only for data X and kernal l are all strictly non negative: ensure inner products are never negative --->capture the intuition X and l are very similar to each other then maybe the inner product between them will be large.

>More esoteric: less often use.
String kernel (use when input is text strings or other types of strings);
chi-square kernel (卡方核函数) ; histogram intersection kernel (直方相交核函数) ;

all kernel funtion /object is to measure similarity of input to landmark.

---

4. Multi-class classification

Many SVM packages already have built-in multi-class classification functionality.
Otherwise use one-vs.-all mehtod:
Have K class: y ε {1, 2, , 3...k}

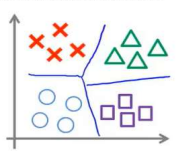
train K SVMs, each one to distinguish one class from the rest:
reset data have only one class and rest -->get paramte θ_1, -->class 1 classfier;

-->get K calssfier with parameter θ_1, θ_2, θ_3, ..θ_K.

pick class i with largest $(\theta_i)^T$ *X

(note: idealy, only for corresponding class i classfire, will get (θ_i)T *X>=0, rest classfire result

Multi-class classification



$y \in \{1, 2, 3, \ldots, K\}$

Many SVM packages already have built-in multi-class classification functionality.
→ Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$
Pick class i with largest $(\theta^{(i)})^T x$ ($y=1$, $y=2$, ... θ of K)

5 Logistic regrssion vs. SVM

SVM , start from logistic regression and modified cost function.

for different regimes as different choice:

1. if n is large (relative to m):
use logistic regression or SVM witout a kernel
e.g: n>=m, n = 10,000, m = 10 , 20, ..1000

in this case many features with small training set, a linear regression probably do fine , as do not have enough parameter/data to fit a very complicated nonlinear function.
(note:
>> training set has 'engough' feature n, -->no need kernel
>> data set: not big-->simplier architecture as linear should be ok)

2. If n is small, m is intermediate:
Use SVM with Gaussian kernel
e.g: n = 1 - 1000, m = 10 - 10,000

(note:
>> training set feature: -->making 'training set' has enough 'feature': by kernel)

3. If n is small, , m is large:
create / add more features, then use logistic regression or SVM without a kernel.
(note: do not use kernel when m is large as computation cost expensive)

e.g: n = 1-1000, , m = 50,000 +
if m is too large, SVM with Gaussin kernel computation willl be slow.

Note: logistic regression and SVM without a kernel is pretty similar: do pretty similar thing and give pretty similar performance. depending on detail one maybe more efficent than another. but if one algorithm is effient for problem, and the other one is to likely to work pretty well as well.

but among the power of SVM with differnt kernel is to learn complex nonlinear functions, having intermediate size training example (m 10,000 - 50,000), feature is also reasonable large n 1-1000, which is common regime, that where SVM with kernel will shine, much better than logistic regression (training example new feature size is transfered to m).

(note: conditons to use logistic regression & SVM:
Depending training set feature number n and volume m:
>>>1. if n is small , and m is intermidate large--> use Gaussin Kernal + SVM
>>>2. Others: n is large / m is too large-kernal is too computation expensive : no dot use kernal -->logistic/SVM is ok
>>>3. if m is very large and n is small-->add/create more feature , enlarge n--> use logistic /SVM

Note:
1. Kernel need to use together with SVM;
2. SVM without kernel is like logistic regression, either one to use is ok. only efficent may be a little different.
SVM should be faster than logistic regression, as cost function much easier to compute gradient??)

3. SVM : motivation: to learn more non-linear decision boundar: get by SVM + kernal

Logistic regression vs. SVMs

n =number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples
→ If n is large (relative to m): (e.g. $n \geq m$, $n = 10{,}000$, $m = 10 \cdots 1000$)
→ Use logistic regression, or SVM without a kernel ("linear kernel")
→ If n is small, m is intermediate: ($n = 1 - 1000$, $m = 10 - 10{,}000$)
→ Use SVM with Gaussian kernel
If n is small, m is large: ($n = 1\text{-}1000$, $m = 50{,}000+$)
→ Create/add more features, then use logistic regression or SVM without a kernel
→ Neural network likely to work well for most of these settings, but may be slower to train.

Summary:

Neural network likely to work well for most of these settings, but may be slower to train.
But if have a good SVM , could run quite a bit faster than neural network.

SVM optimizaiton problem: is convex.
Good SVM package will find the global minimum or something close to it.: (note: due to well designed kernel?)
for SVM do not need to worry about local optima. (In practice, local optima is not big problem for neural network either-saddle point, -with momentum/RMS/Adam, but using SVM have one less thing to worry about)

about to choose which algorithm:

algorithm itself does matter, but what matter even more is things like how much **data** have, how skilled / good are you at doing **error analysis**, and **deguging** learning algorithms, firguring out how to design new **features** and figure out what other features to give algorithm.
often those things will matter more than using logistic regression or an SVM.

Together with logistic regrssion , SVM, neural network, using those to speed up learing algorithm, are very well positivioned to build state of the art, machine learning systems for a wide region for applications. this is another powerful tool to have in your arsenal.

Summary

SVM:

COST FUNCTON/ Object: modified based on logistic cost function:

prediction fucntion: is linear regression: y=1 when θ_0 + θ_1 *x >=0

(note: linear function: h_θ(x)=θ_0 + θ_1 *x)

process: based on training data:

1. choose algorithm model
>> linear: h_θ(x)=θ_0 + θ_1 *x; (1)
>> logistic: h_θ(x)= g(θ_0 + θ_1 *x); (2)--->Logistic regression
>> SVM: h_θ(x)=1, when θ_0 + θ_1 *x>=0; (3)二分类

>1. Linear regeression: y is continue value: y= h_θ(x)_formula 1;
>2. Logistic regression (for classification problem): h_θ(x)_formula 2, output is possibility: 0-1 range for being the class
>3. SVM (for classficaiton problem) : formula 3: output is 1 / 0 , acc.to θ_0 + θ_1 *x>=0;

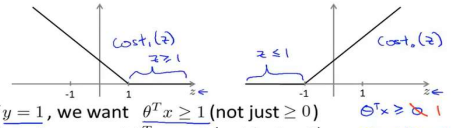
2. compute Cost function:

>1. Linear regeression: minimizing squared error: distane of predition and labled value
>2. Logistic regression: log function; max probability of being the label class
>3. SVM : linear line + 0.; maximizing training set distance to decision boundary

3. Fining algorithm parameter by min cost function

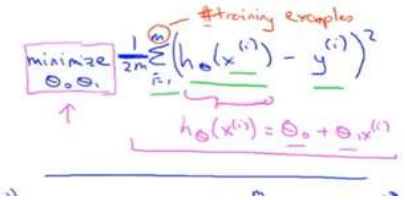
Support Vector Machine

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



→ 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)
C = 100,000

Andrew Ng



Alternative view of logistic regression

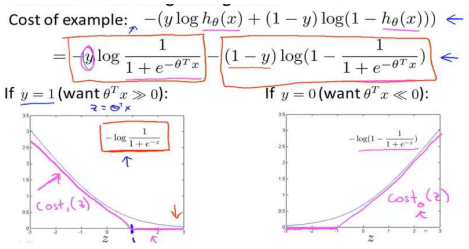
Grdient descent or numerical formula, or advanced algorithm

4. Output:

>1. Linear regeression: value
>2. Logistic regression: possibility
>3. SVM : 0/1

4. Input:

>1. Linear regeression: x (better scal feature first)
>2. Logistic regression: X + new feature (scal feature first _for gradient)
>3. SVM : with kernel: new feature representing :similarity to landmark/input x. scal input x first, then generate new feature f(i)

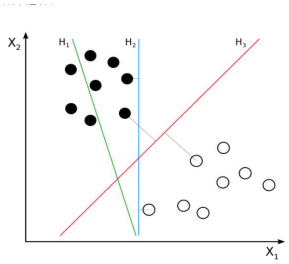


Understanding SVM:

[看了这篇文章你还不懂SVM你来打我完整搬运过来的]

[我是这样理解--SVM，不需要繁杂公式的那种！(附代码)]

SVM就是一种二类分类模型，他的基本模型是的定义在特征空间上的间隔最大的线性分类器，SVM的 note: decision boundary: θT*x =0;
margin: when C is large: margin = distance of θT*x =1 & θT*x =-1-->d = 2/|| θ||



- 线性可分SVM

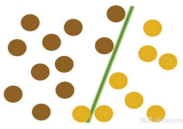
> 当训练数据线性可分时，通过硬间隔(hard margin，什么是硬、软间隔下面会讲)最大化可以学习得到一个线性分类器，即硬间隔SVM，如上图的的H3。

- 线性SVM

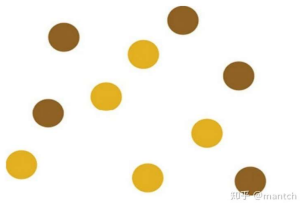
> 当训练数据不能线性可分但是可以近似线性可分时，通过软间隔(soft margin)最大化也可以学习到一个线性分类器，即软间隔SVM。

- 非线性SVM

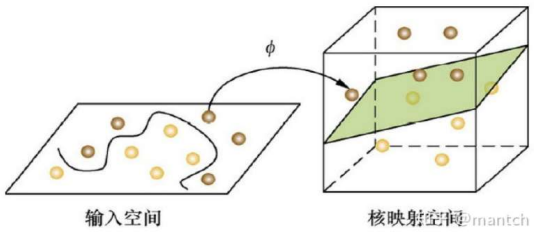
> 当训练数据线性不可分时，通过使用核技巧(kernel trick)和软间隔最大化，可以学习到一个非线性SVM。



SVM实际上是在为天使找到木棒的最佳放置位置，使得两边的球都离分隔它们的木棒足够远。依照SVM为天使选择的木棒位置，魔鬼即使按刚才的方式继续加入新球，木棒也能很好地将两类不同的球分开。



按照这种球的摆法，世界上貌似没有一根木棒可以将它们 完美分开。但天使毕竟有法力，他一拍桌子，便让这些球飞到了空中，然后凭借 念力抓起一张纸片，插在了两类球的中间。从魔鬼的角度看这些 球，则像是被一条曲线完美的切开了。



后来，“无聊”的科学家们把这些球称为“数据”，把木棍称为“分类面”，找到最 大间隔的木棒位置的过程称为“优化”，拍桌子让球飞到空中的念力叫“核映射”，在 空中分隔球的纸片称为“分类超平面”。这便是SVM的童话故事。

Cost function: min: C * A + B:

--> min B , max distance of support vector (class-1) and class(0): margin: when C is large: margin = distance of θT*x =1 & θT*x =-1-->d = 2/|| θ||

-->A: cost loss for the outlier

-->C: penalizing factor for the outlier cost loss.

2.2 间隔最大化

将高数里面求两条平行直线的距离公式推广到高维可求得图2.1中margin的 ρ:

$$margin = \rho = \frac{2}{||W||} \tag{2.2.1}$$

我们的目标是使 ρ 最大, 等价于使 ρ^2 最大:

$$\underset{W,b}{max}\rho \iff \underset{W,b}{max}\rho^2 \iff \underset{W,b}{min}\frac{1}{2}||W||^2 \tag{2.2.2}$$

上式的 $\frac{1}{2}$ 是为了后续求导后刚好能消去，没有其他特殊意义。

同时也不要忘了有一些约束条件:

$$X_i^T W + b \geq +1, y_i = +1 \tag{2.2.3}$$
$$X_i^T W + b \leq -1, y_i = -1$$

总结一下，间隔最大化问题的数学表达就是

$$\underset{W,b}{min}J(W) = \underset{W,b}{min}\frac{1}{2}||W||^2 \tag{2.2.4}$$
$$s.t. \quad y_i(X_i^T W + b) \geq 1, i = 1, 2, \ldots n.$$

通过求解上式即可得到最优超平面 $\hat{W}$ 和 $\hat{b}$。具体如何求解见2.4和2.5节。

2.3 支持向量

在线性可分的情况下，训练数据集的样本点中与分离超平面距离最近的数据点称为支持向量(support vector)，支持向量是使 $(2.2.4)$ 中的约束条件取等的点，即满足

$$y_i(X_i^T W + b) = 1 \tag{2.3.1}$$

的点。也即所有在直线 $X^T W + b = 1$ 或直线 $X^T W + b = -1$ 的点。如下图所示:

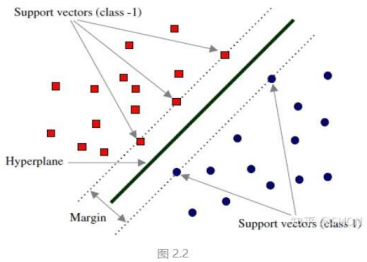


图 2.2

在决定最佳超平面时只有支持向量起作用，而其他数据点并不起作用(具体推导见2.4节最后)。如果移动非支持向量，甚至删除非支持向量都不会对最优超平面产生任何影响。也即支持向量对模型起着决定性的作用，这也是“支持向量机”名称的由来。

Note: if remove other samples not in support vectors-->θT*x(i) >=1: parameter θ may change-->decision boundary θT*x=0 change

3. 线性SVM——软间隔

在前面的讨论中，我们一直假定训练数据是严格线性可分的，即存在一个超平面能完全将两类数据分开。但是现实任务这个假设往往不成立，例如下图所示的数据。

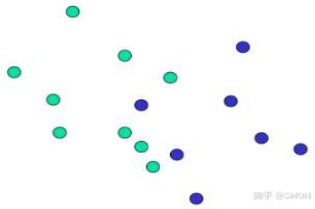


图3.1

3.1 软间隔最大化

解决该问题的一个办法是允许SVM在少量样本上出错，即将之前的硬间隔最大化条件放宽一点，为此引入“软间隔(soft margin)”的概念，即允许少量样本不满足约束

$$y_i(X_i^T W + b) \geq 1 \tag{3.1.1}$$

为了使不满足上述条件的样本点尽可能少，我们需要在优化的目标函数 $(2.2.2)$ 里面新增一个对这些点的惩罚项。最常用的是hinge损失:

$$l_{hinge}(z) = max(0, 1-z) \tag{3.1.2}$$

即若样本点满足约束条件损失就是0, 否则损失就是 $1-z$,则优化目标 $(2.2.2)$ 变成

$$\min_{W,b} \quad \frac{1}{2}||W||^2 + C\sum_{i=1}^{n} max(0, 1-y_i(X_i^T W + b)) \tag{3.1.3}$$

其中 $C > 0$ 称为惩罚参数，C 越小时对误分类惩罚越小，越大时对误分类惩罚越大，当 C 取正无穷时就变成了硬间隔优化。实际应用时我们要合理选取 C，C 越小越容易欠拟合，C 越大越容易过拟合。

如果我们引入“松弛变量” $\xi_i \geq 0$, 那么式 $(3.1.3)$ 可重写成

$$\begin{aligned} \min_{W,b,\xi} \quad & \frac{1}{2}||W||^2 + C\sum_{i=1}^{n} \xi_i \\ s.t. \ & y_i(X_i^T W + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, 2, \dots n. \end{aligned} \tag{3.1.4}$$

上式所述问题即软间隔支持向量机。

3.3 惩罚参数 C

对于不同惩罚参数 C，SVM结果如下图所示

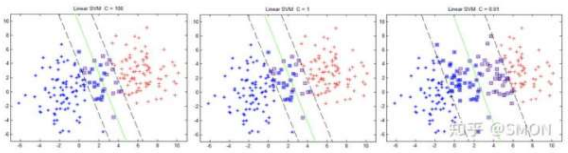


图 3.3

再来看看我们的原始目标函数:

$$\min_{W,b,\xi} \quad \frac{1}{2}||W||^2 + C\sum_{i=1}^{n} \xi_i$$

对于更加一般化的问题，可将上述式子抽象成:

$$\min_{f} \quad \Omega(f) + C\sum_{i=1}^{n} l(f(x_i), y_i) \tag{3.3.1}$$

前一项可以理解为“结构风险(structural risk)”，用来描述所求模型的某些性质(SVM就是要求间隔最大)；第二项称为“经验风险(empirical risk)”，用来描述模型与训练数据的契合程度(即误差)。而参数 C 就是用于对二者的折中,即我们一方面要求模型要满足某种性质另一方面又想使模型与训练数据很契合。

从正则化角度来讲，$\Omega(f)$ 称为正则化项，C 称为惩罚参数，C 越大即对误分类的惩罚越大(要求模型对训练模型更契合)，这可能会存在过拟合；C 越小即相对更加看重正则化项，此时可能存在欠拟合。

4.1 核函数

如下图所示，核技巧的基本思路分为两步:*使用一个变换将原空间的数据映射到新空间(例如更高维甚至无穷维的空间)；*然后在新空间里用线性方法从训练数据中学习得到模型。

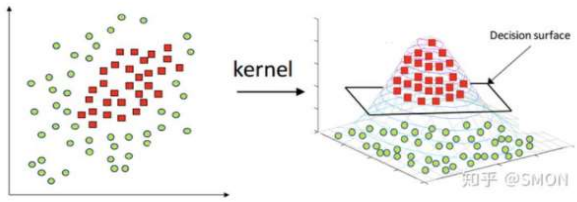


图 4.1

怎样映射到特征空间?

先来看看核函数的定义，

> 设 $\mathcal{X}$ 是输入空间(欧式空间 R^n 的子集或离散集合)，又设 $\mathcal{H}$ 是特征空间(希尔伯特空间)，如果存在一个 $\mathcal{X}$ 到 $\mathcal{H}$ 的映射 $\phi(x): \mathcal{X} \to \mathcal{H}$ 使得对所有 $x, z \in \mathcal{X}$，函数 $K(x, z)$ 满足条件 $K(x, z) = \phi(x) \cdot \phi(z)$ 则称 $K(x, z)$ 为核函数，$\phi(x)$ 为映射函数，式中 $\phi(x) \cdot \phi(z)$ 为

4.2 正定核

由上面的介绍可知，我们只需要定义核函数就可以了。但是如何不通过映射 $\phi(x)$ 判断给定的一个函数 $K(x, z)$ 是不是核函数呢？或者说，$K(x, z)$ 需要满足什么条件才是一个核函数。

通常所说的核函数就是正定核函数，下面不加证明的给出正定核的充要条件，具体证明略显复杂，有兴趣的可以参考《统计学习方法》。

> 设 $\mathcal{X} \subset R^n$, $K(x, z)$ 是定义在 $\mathcal{X} \times \mathcal{X}$ 上的对称函数，如果对任意的 $x_i \in \mathcal{X}, i = 1, 2, \dots, m$，$K(x, z)$ 对应的Gram矩阵 $K = [K(x_i, x_j)]_{m \times m}$ 是半正定矩阵，则 $K(x, z)$ 是正定核。

虽然有了上述定义，但是实际应用时验证 $K(x, z)$ 是否是正定核依然不容易，因此在实际问题中一般使用已有的核函数，下面给出一些常用的核函数。

- 多项式核函数(polynomial kernel function)

$$K(x, z) = (x \cdot z + 1)^p \tag{4.2.1}$$

- 高斯核函数(Guassian kernel function)

$$K(x, z) = exp(-\frac{||x-z||^2}{2\sigma^2}) \tag{4.2.2}$$

$\phi(x)$ 和 $\phi(z)$ 的内积。

通常，直接计算 $K(x,z)$ 比较容易而通过 $\phi(x)$ 和 $\phi(z)$ 计算 $K(x,z)$ 并不容易。而幸运的是，在线性支持向量机的对偶问题中，无论是目标函数还是决策函数都只涉及到输入样本与样本之间的内积，因此我们不需要显式地定义映射 $\phi(x)$ 是什么而只需事先定义核函数 $K(x,z)$ 即可。也就是说，在核函数 $K(x,z)$ 给定的情况下，可以利用解线性问题的方法求解非线性问题的支持向量机，此过程是隐式地在特征空间中进行的。

4.3 非线性支持向量机

如前4.1、4.2所述，利用核技巧可以很简单地把线性支持向量机扩展到非线性支持向量机，只需将线性支持向量机中的内积换成核函数即可。下面简述非线性支持向量机学习算法。

- 首先选取适当的核函数 $K(x,z)$ 和适当的参数 C，构造最优化问题

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K(X_i,X_j) - \sum_{i=1}^{n}\alpha_i \\ s.t. \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \\ & 0 \le \alpha_i \le C, i = 1,2,\dots,n. \end{aligned} \tag{4.3.1}$$

- 再利用现成的二次规划问题求解算法或者SMO算法求得最优解 $\hat{\alpha}$。
- 选择 $\hat{\alpha}$ 的一个满足 $0 < \hat{\alpha}_j < C$ 的分量 $\hat{\alpha}_j$，计算

$$\hat{b} = y_j - \sum_{i \in SV}\hat{\alpha}_i y_i K(X_j, X_i) \tag{4.3.2}$$

- 构造决策函数：

$$f(x) = sign(\sum_{i \in SV}\hat{\alpha}_i y_i K(X_j, X_i) + \hat{b}) \tag{4.3.3}$$

5. 总结

任何算法都有其优缺点，支持向量机也不例外。

支持向量机的优点是:

1. 由于SVM是一个凸优化问题，所以求得的解一定是全局最优而不是局部最优。
2. 不仅适用于线性线性问题还适用于非线性问题(用核技巧)。
3. 拥有高维样本空间的数据也能用SVM，这是因为数据集的复杂度只取决于支持向量而不是数据集的维度，这在某种意义上避免了“维数灾难”。
4. 理论基础比较完善(例如神经网络就更像一个黑盒子)。

支持向量机的缺点是:

1. 二次规划问题求解将涉及m阶矩阵的计算(m为样本的个数), 因此SVM不适用于超大数据集。(SMO算法可以缓解这个问题)
2. 只适用于二分类问题。(SVM的推广SVR也适用于回归问题；可以通过多个SVM的组合来解决多分类问题)

机器学习中SVD和PCA一直没有搞的特别清楚，应该如何理解呢?

石溪，清华大学 计算机科学与技术硕士

这篇回答节选自我在专栏《机器学习中的数学：线性代数》中的一篇文章，我们一起来谈谈奇异值分解的来龙去脉。

欢迎关注我的知乎账号 @石溪，将持续发布机器学习数学基础及算法应用等方面的精彩内容。

1.再谈特征值分解的几何意义

在介绍奇异值分解（SVD）之前，我们先回顾一下特征值分解的几何意义。

1.1.分解过程回顾

我们最开始获得的是一组原始的 $m \times n$ 数据样本矩阵 A，其中，m 表示特征的个数，n 表示样本的个数。通过与自身转置相乘：AA^T 得到了样本特征的 m 阶协方差矩阵 C，通过求取协方差矩阵 C 的一组标准正交特征向量 $q_1, q_2, \dots q_m$ 以及对应的特征值 $\lambda_1, \lambda_2, \dots, \lambda_m$。

我们这里处理的就是协方差矩阵 C，对 C 进行特征值分解，将矩阵分解成了

$$C = \begin{bmatrix} q_1 & q_2 & \dots & q_m \end{bmatrix} \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \dots & \\ & & & \lambda_m \end{bmatrix} \begin{bmatrix} q_1^T \\ q_2^T \\ \dots \\ q_m^T \end{bmatrix}。$$

最终，我们选取前 k 个特征向量构成数据压缩矩阵 P 的各行，通过 PA 达到数据压缩的目的。

2.从 $Av = \sigma u$ 入手奇异值分解

但是，如果我们不进行协方差矩阵 C 的求取，绕开它直接对原始的数据采样矩阵 A 进行矩阵分解，从而进行降维操作，行不行？

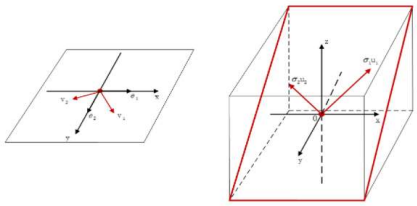
如果继续沿用上面的办法，显然是不行的，特征值分解对矩阵的要求很严，首先得是一个方阵，其次在方阵的基础上，还得满足可对角化的要求。但是原始的 $m \times n$ 数据采样矩阵 A 连方阵这个最基本的条件都不满足，是根本无法进行特征值分解的。

找不到类似 $Ap = \lambda p$ 的核心等式了，当不是无能为力了？怎料，天无绝人之路，这里，我首先给大家介绍一个对于任意 $m \times n$ 矩阵的更具普遍意义的一般性质：

对于一个 $m \times n$，秩为r的矩阵 A，这里我们暂且假设 $m > n$，于是就有 $r \le n < m$ 的不等关系。我们在 R^n 空间中一定可以找到一组标准正交向量 $v_1, v_2, \dots v_n$，在 R^m 空间中一定可以找到另一组标准正交向量 $u_1, u_2, \dots, u_m$，使之满足 n 组相等关系：$Av_i = \sigma_i u_i$，其中（i 取 $1 \sim n$）。

$Av_i = \sigma_i u_i$，这个等式非常神奇，我们仔细的揭开里面的迷雾，展现他的精彩之处：

矩阵 A 是一个 $m \times n$ 的矩阵，他所表示的线性变换是将 n 维原空间中的向量映射到更高维的 m 维目标空间中，而 $Av_i = \sigma_i u_i$ 这个等式意味着，在原空间中找到一组新的标准正交向量 $\begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$，在目标空间中存在着对应的一组标准正交向量 $\begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix}$，此时 v_i 与 u_i 线性无关。当矩阵 A 作用在原空间上的某个基向量 v_i 上时，其线性变换的结果就是：对应在目标空间中的 u_i 向量沿着自身方向伸长 σ_i 倍，并且任意一对 (v_i, u_i) 向量都满足这种关系（显然特征值分解是这里的一种特殊情况，即两组标准正交基向量相等）。如图2所示：



在 $Av_i = \sigma_i u_i$ 的基础上，我们明白该如何往下走了：

$A \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix} = \begin{bmatrix} \sigma_1 u_1 & \sigma_2 u_2 & \dots & \sigma_n u_n \end{bmatrix}$，转换成：

$$A \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \dots & \\ & & & \sigma_n \end{bmatrix}。$$

2. 数据降维

降维就是一种对高维度特征数据预处理方法。降维是将高维度的数据保留下最重要的一些特征，去除噪声和不重要的特征，从而实现提升数据处理速度的目的。在实际的生产和应用中，降维在一定的信息损失范围内，可以为我们节省大量的时间和成本。降维也成为应用非常广泛的数据预处理方法。

降维具有如下一些优点：

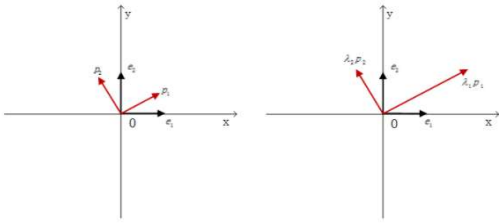
- 使得数据集更易使用。
- 降低算法的计算开销。
- 去除噪声。
- 使得结果容易理解。

降维的算法有很多，比如奇异值分解(SVD)、主成分分析(PCA)、因子分析(FA)、独立成分分析(ICA)。

1.2.几何意义剖析

以上是回顾之前的内容，不难发现，为了完成矩阵的特征值分解，最最关键还是要回归到这个基本性质上来：$Cq_i = \lambda_i q_i$。

我们为什么又提这个呢？结合主成分分析的推导过程我们知道，协方差矩阵 C 之所以能够分解，是因为在原始空间 R^n 中，我们原本默认是用 $e_1, e_2, \dots, e_m$ 这组默认基向量来表示我们空间中的任意一个向量 a，如果我们采用基变换，将 a 用 $q_1, q_2, \dots, q_m$ 这组标准正交基来表示后，Ca 的乘法运算就变得很简单了，只需要在各个基向量的方向上对应伸长 λ_i 倍即可，如图1所示：



实际上，我们之前也重点分析过，因为协方差矩阵具备对称性、正定性，保证了他可以被对角化，并且特征值一定为正，从而使得特征值分解的过程一定能够顺利完成。

因此利用特征值分解进行主成分分析，核心就是获取协方差矩阵，然后对其进行矩阵分解，获得一组特征值和其对应的方向。

3.着手尝试分解

此时感觉还差一点，因为我们发现 $u_{n+1}, u_{n+2}, \dots, u_m$ 并没有包含在式子里，我们把他们加进去，把 $u_{n+1}, u_{n+2}, \dots, u_m$ 加到矩阵右侧，形成完整的 m 阶方阵 $U = \begin{bmatrix} u_1 & u_2 & \dots & u_n & u_{n+1} & \dots & u_m \end{bmatrix}$，在对角矩阵下方加上 $m - n$ 个全零行，形成 $m \times n$ 的矩阵 $\Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \dots & \\ & & & \sigma_n \\ & & & \end{bmatrix}$，很明显由于 Σ 矩阵最下面的 $m - n$ 行全零，因此右侧的计算结果不变，等式依然成立。

此时就有：$AV = U\Sigma$，由于 V 的各列是标准正交向量，因此 $V^{-1} = V^T$，移到等式右侧，得到了一个矩阵分解的式子：

$A = U\Sigma V^T$，其中 U 和 V 是由标准正交向量构成的 m 阶和 n 阶方阵，而 Σ 是一个 $m \times n$ 的对角矩阵（注意，不是方阵）。

4.分析分解过程中的细节

从大的框架宏观来看，这个结论非常漂亮，在原空间和目标空间中各找一组标准正交基，就轻轻松松的把对角化的一系列要求轻松化解。直接得到了数据采样矩阵 A 的矩阵分解形式。

但是此时还有一个最关键的地方似乎没有明确：就是方阵 U 和方阵 V 该如何取得，以及 Σ 矩阵中的各个值应该为多少，我们借助在对称矩阵那一节中储备的基础知识来——化解。

我们还是从 $A = U\Sigma V^T$ 式子入手。首先，获取转置矩阵 $A^T = (U\Sigma V^T)^T = V\Sigma U^T$，我们在此基础上可以获取两个对称矩阵：

第一个对称矩阵是：$A^T A = V\Sigma U^T U\Sigma V^T$，由于 U 的各列是标准正交向量，因此 $U^T U = I$，式子最终化简为了：$A^T A = V\Sigma^2 V^T$。

同理，第二个对称矩阵：$AA^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T$。

这里我们结合对称矩阵那一节中的一个重要结论，揭示一下这里面的所有细节：$A^T A$ 是 n 阶对称方阵，AA^T 是 m 阶对称方阵。他们的秩相等，为 $r = Rank(A)$。因此他们拥有完全相同的 r 个非零特征值，从大到小排列为：$\lambda_1, \lambda_2, \dots \lambda_r$，两个对称矩阵的剩余 $n - r$ 个和 $m - r$ 个特征值为 0。这进一步也印证了 $A^T A = V\Sigma^2 V^T$ 和 $AA^T = U\Sigma^2 U^T$ 对角线上的非零特征值是完全一样的。

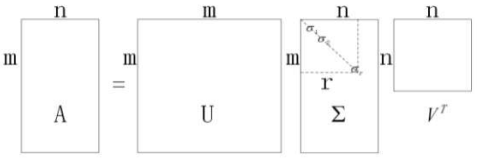
同时，由对称矩阵的性质可知：$A^T A$ 一定含有 n 个标准正交特征向量，对应特征值从大到小的顺序排列为：$\begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$，而 AA^T 也一定含有 m 个标准正交特征向量，对应特征值从大到小依次排列为 $\begin{bmatrix} u_1 & u_2 & \dots & u_m \end{bmatrix}$。这里的 v_i 和 u_i ——对应。

对应的 Σ 矩阵也很好求，求出 AA^T 或 $A^T A$ 的非零特征值，从大到小排列为：$\lambda_1, \lambda_2, \dots, \lambda_r$，$\Sigma$ 矩阵中对角线上的非零值 σ_i 则依次为：$\sqrt{\lambda_1}$，$\sqrt{\lambda_2}, \dots, \sqrt{\lambda_r}$。因此，$\Sigma$ 矩阵对角线上 σ_r 以后的元素均为 0 了。

整个推导分析过程结束，我们隐去零特征值，最终得到了最完美的SVD分解结果：

$A = \begin{bmatrix} u_1 & u_2 & \dots & u_m \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \dots & \\ & & & \sigma_r \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \dots \\ v_n^T \end{bmatrix}$，这里 $r \le n < m$。

我们用一个抽象图来示意一下分解的结果，有助于大家加深印象，如图3所示：



由此，我们顺利的得到了任意 $m \times n$ 矩阵 A 的SVD分解形式。

3. PCA原理详解

3.1 PCA的概念

PCA(Principal Component Analysis)，即主成分分析方法，是一种使用最广泛的数据降维算法。PCA的主要思想是将n维特征映射到k维上，这k维是全新的正交特征也被称为主成分，是在原有n维特征的基础上重新构造出来的k维特征。PCA的工作就是从原始的空间中顺序地找一组相互正交的坐标轴，新的坐标轴的选择与数据本身是密切相关的。其中，第一个新坐标轴选择是原始数据中方差最大的方向，第二个新坐标轴选取是与第一个坐标轴正交的平面中使得方差最大的，第三个轴是与第1,2个轴正交的平面中方差最大的。依次类推，可以得到n个这样的坐标轴。通过这种方式获得的新的坐标轴，我们发现，大部分方差都包含在前面k个坐标轴中，后面的坐标轴所含的方差几乎为0。于是，我们可以忽略余下的坐标轴，只保留前面k个含有绝大部分方差的坐标轴。事实上，这相当于只保留包含绝大部分方差的维度特征，而忽略包含方差几乎为0的特征维度，实现对数据特征的降维处理。

思考：我们如何得到这些包含最大差异性的主成分方向呢?

答案：事实上，通过计算数据矩阵的协方差矩阵，然后得到协方差矩阵的特征值特征向量，选择特征值最大(即方差最大)的k个特征所对应的特征向量组成的矩阵。这样就可以将数据矩阵转换到新的空间当中，实现数据特征的降维。

由于得到协方差矩阵的特征值特征向量有两种方法：特征值分解协方差矩阵、奇异值分解协方差矩阵，所以PCA算法有两种实现方法：基于特征值分解协方差矩阵实现PCA算法、基于SVD分解协方差矩阵实现PCA算法。

既然提到协方差矩阵，那么就简单介绍一下方差和协方差的关系。然后概括介绍一下特征值分解矩阵原理、奇异值分解矩阵的原理。概括介绍是因为在我之前的《机器学习中SVD总结》文章中已经

3.2 协方差和散度矩阵

样本均值：

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{N} x_i$$

样本方差：

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

样本X和样本Y的协方差：

$$\begin{aligned} Cov(X,Y) &= E\left[(X - E(X))(Y - E(Y))\right] \\ &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \end{aligned}$$

由上面的公式，我们可以得到以下结论：

(1) 方差的计算公式是针对一维特征，即针对同一特征不同样本的取值来进行计算得到；而协方差则必须要求至少满足二维特征；方差是协方差的特殊情况。

(2) 方差和协方差的除数是n-1,这是为了得到方差和协方差的无偏估计。

协方差为正时，说明X和Y是正相关关系；协方差为负时，说明X和Y是负相关关系；协方差为0时，说明X和Y是相互独立。Cov(X,X)就是X的方差。当样本是n维数据时，它们的协方差实际上是协方差矩阵(对称方阵)。例如，对于3维数据(x,y,z)，计算它的协方差就是：

$$Cov(X,Y,Z) = \begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

散度矩阵定义为：

The scatter matrix is computed by the following equation:

$$S = \sum_{k=1}^{n}(\boldsymbol{x}_k - \boldsymbol{m})(\boldsymbol{x}_k - \boldsymbol{m})^T$$

where $\boldsymbol{m}$ is the mean vector

$$\boldsymbol{m} = \frac{1}{n}\sum_{k=1}^{n}\boldsymbol{x}_k$$

知乎 @Microstrong

散度矩阵

对于数据X的散度矩阵为 XX^T 。其实协方差矩阵和散度矩阵关系密切，散度矩阵就是协方差矩阵乘以（总数据量-1）。因此它们的**特征值**和**特征向量**是一样的。这里值得注意的是，散度矩阵是**SVD奇异值分解**的一步，因此PCA和SVD是有很大联系。

3.3 特征值分解矩阵原理

(1) 特征值与特征向量

如果一个向量v是矩阵A的特征向量，将一定可以表示成下面的形式：

$$Av = \lambda v$$

其中，λ是特征向量v对应的特征值，一个矩阵的一组特征向量是一组正交向量。

(2) 特征值分解矩阵

对于矩阵A，有一组特征向量v，将这组向量进行正交化单位化，就能得到一组正交单位向量。**特征值分解**，就是将矩阵A分解为如下式：

$$A = Q\Sigma Q^{-1}$$

其中，Q是矩阵A的特征向量组成的矩阵，Σ则是一个对角阵，对角线上的元素就是特征值。

具体了解这一部分内容看我的《机器学习中SVD总结》文章。地址：机器学习中SVD总结

3.4 SVD分解矩阵原理

奇异值分解是一个能适用于任意矩阵的一种分解的方法，对于任意矩阵A总是存在一个奇异值分解：

$$A = U\Sigma V^T$$

假设A是一个m*n的矩阵，那么得到的U是一个m*m的方阵，U里面的正交向量被称为左奇异向量。Σ是一个m*n的矩阵，Σ除了对角线其它元素都为0，对角线上的元素称为奇异值。 V^T 是v的转置矩阵，是一个n*n的矩阵，它里面的正交向量被称为右奇异值向量。而且一般来讲，我们会将Σ上的值按从大到小的顺序排列。

SVD分解矩阵A的步骤：

(1) 求 AA^T 的特征值和特征向量，用单位化的特征向量构成 U。

(2) 求 A^TA 的特征值和特征向量，用单位化的特征向量构成 V。

(3) 将 AA^T 或者 A^TA 的特征值求平方根，然后构成 Σ。

具体了解这一部分内容看我的《机器学习中SVD总结》文章。地址：机器学习中SVD总结

3.5 PCA算法两种实现方法

(1) 基于特征值分解协方差矩阵实现PCA算法

输入：数据集 $X = \{x_1, x_2, x_3, \ldots, x_n\}$ ，需要降到k维。

1) 去平均值(即去中心化)，即每一位特征减去各自的平均值。

2) 计算协方差矩阵 $\frac{1}{n}XX^T$,注：这里除或不除样本数量n或n-1,其实对求出的特征向量没有影响。

3) 用特征值分解方法求协方差矩阵 $\frac{1}{n}XX^T$ 的特征值与特征向量。

4) 对特征值从大到小排序，选择其中最大的k个。然后将其对应的k个特征向量分别作为行向量组成特征向量矩阵P。

5) 将数据转换到k个特征向量构建的新空间中，即Y=PX。

总结：

1)关于这一部分为什么用 $\frac{1}{n}XX^T$,这里面含有很复杂的线性代数理论推导，想了解具体细节的可以看下面这篇文章。

CodingLabs - PCA的数学原理

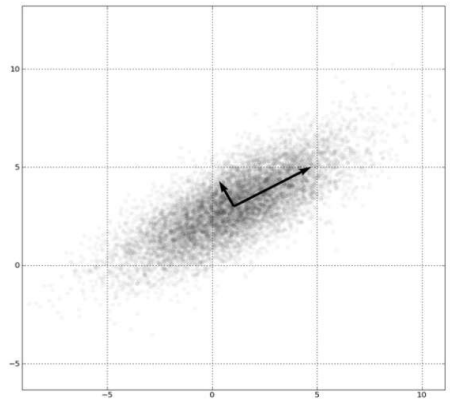
2)关于为什么用特征值分解矩阵，是因为 $\frac{1}{n}XX^T$ 是方阵，能很轻松的求出特征值与特征向量。

当然，用奇异值分解也可以，是求特征值与特征向量的另一种方法。

PCA的数学原理

如下图，平面上有很多二维空间的特征点，如果想对这些特征点做特征降维（变为一维），应该怎么做呢？大家应该都知道需要进行投影，但还要考虑在哪个方向上进行投影，例如图中需要投影到长箭头方向即可，但考虑为什么不在短箭头上投影？

PCA本质上是一个**有损的特征压缩过程**，但是我们期望损失的精度尽可能地少，也就是希望压缩的过程中保留最多的原始信息。要达到这种目的，我们希望降维（投影）后的数据点尽可能地分散。如图，相比于长箭头，如果在短箭头上进行投影，那么重叠的点会更多，也就意味着信息丢失的更多，因而选择长箭头方向。



基于这种思想，我们希望投影后的数据点尽可能地分散。而这种分散程度在数学上可以利用方差来表示。设降维后的特征为 A，也就是希望 $var(A)=\frac{1}{m}\sum_i^m(a_i-\mu_a)^2$ 尽可能地大（a_i 为特征 A 中的值，μ_a 为均值），而由于在PCA降维前，一般已经做了**特征零均值化**处理，为了方便，记 $var(A)=\frac{1}{m}\sum_i^m a_i^2$。

同样，为了减少特征的冗余信息，我们希望降维后的各特征之间互不相关。而不相关性可以用协方差来衡量。设降维后的两个特征为 A、B，则希望 $Cov(A,B)=\frac{1}{m}\sum_i^m a_ib_i$ 为0。

现假设我们的数据为

$$X=\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}$$

PCA的推导证明

PCA的构建：PCA需要构建一个编码器 f，由输入 $x\in R^n$ 得到一个最优编码 $c\in R^l$（若 $l<n$，则做了降维编码）；同时有一个解码器 g，解码后的输出 $g(c)$ 尽可能地与 x 相近。

PCA由我们所选择的解码器决定，在数学上，我们使用矩阵将 c 映射回 R^n，即 $g(c)=Dc$，其中 $D\in R^{n\times l}$ 定义解码的矩阵。

为了限制PCA的唯一性，我们限制 D 中所有列向量彼此正交且均有单位范数（否则 D、c 同比例增加、减少会产生无数个解）。

在数学上，为了满足PCA构建中的条件，我们利用 L_2 范数来衡量 $g(c)$ 与 x 的相近程度。即 $c^*=argmin_c||x-g(c)||_2$，也就是 $c^*=argmin_c||x-g(c)||_2^2$

该最小化函数可以简化为

$$\begin{aligned}&(x-g(c))^T(x-g(c))\\&=x^Tx-x^Tg(c)-g(c)^Tx+g(c)^Tg(c)\\&=x^Tx-2x^Tg(c)+g(c)^Tg(c)\end{aligned}$$

因而，优化目标变为 $c^*=argmin_c-2x^Tg(c)+g(c)^Tg(c)$，再带入 $g(c)=Dc$，

$$\begin{aligned}c^*&=argmin_c-2x^TDc+c^TD^TDc\\&=argmin_c-2x^TDc+c^Tc(D^TD=I_l)\end{aligned}$$

再求偏导

$$\begin{aligned}&\nabla_c(-2x^TDc+c^Tc)=0\\&-2D^Tx+2c=0\\&c=D^Tx\end{aligned}$$

于是我们可以得到编码函数 $f(x)=D^Tx$，PCA的重构操作也就可以定义为 $r(x)=g(c)=g(f(x))=DD^Tx$。问题接着就转化成如何求编码矩阵 D。由于PCA算法是在整个数据矩阵上进行编码，因而也要用 D 对所有数据进行解码，所以需要最小化所有维上的误差矩阵的Frobenius范数：

$$D^*=argmin_D\sqrt{\sum_{i,j}(x_j^{(i)}-r(x^{(i)}))_j^2}\ \ subject\ to\ D^TD=I_l$$

我们考虑 $l=1$ 的情况，则 D 是一个单一向量 d，则上式可以转化为

$$d^*=argmin_d\sum_i||(x^{(i)}-ddx^{(i)}||_2^2\ \ subject\ to\ ||d||_2=1$$

而 $d^Tx^{(i)}$ 为标量，转置与自身相等，上式通常写作

$$X = \begin{bmatrix} \vdots & \vdots \\ a_m & b_m \end{bmatrix}$$

构造出协方差矩阵，并乘以系数 $\frac{1}{m}$，则

$$\frac{1}{m}X^TX = \begin{bmatrix} \frac{1}{m}\sum_i^m a_i^2 & \frac{1}{m}\sum_i^m a_i b_i \\ \frac{1}{m}\sum_i^m a_i b_i & \frac{1}{m}\sum_i^m b_i^2 \end{bmatrix}$$

可以看出 $\frac{1}{m}X^TX$ 的对角线元素就是各特征的方差，其他各位置的元素就是各特征之间的协方差。因而只需要降维后的数据协方差矩阵满足对角矩阵的条件即可。

设 Y 为原始数据 X 做完PCA降维后的数据，满足 $Y = XP$（矩阵乘法相当于映射，若 P 为的列向量为基向量，那么就相当于映射到新的坐标系），Y_c，X_c 分别为对应的协方差矩阵，那么

$$\begin{aligned} Y_c &= \frac{1}{m}Y^TY \\ &= \frac{1}{m}(XP)^TXP \\ &= \frac{1}{m}P^TX^TXP \\ &= P^T(\frac{1}{m}X^TX)P \\ &= P^TX_cP \end{aligned}$$

因而，我们只需要计算出 P，使 $Y_c = P^TX_cP$ 满足对角矩阵的条件即可。而 X_c 为实对称矩阵，我们只需要对它做矩阵对角化即可。

PCA的原理基本就是这样，还是挺简单的。

1.PCA降维的计算过程

下图是从西瓜书里截取的PCA降维过程的图片。

输入： 样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$；
低维空间维数 d'.

过程：

1: 对所有样本进行中心化：$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \frac{1}{m}\sum_{i=1}^m \boldsymbol{x}_i$;
2: 计算样本的协方差矩阵 $\mathbf{X}\mathbf{X}^{\mathrm{T}}$;
3: 对协方差矩阵 $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ 做特征值分解;
4: 取最大的 d' 个特征值所对应的特征向量 $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{d'}$.

输出： 投影矩阵 $\mathbf{W} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{d'})$.

需要说明的是，算法中的向量为列向量。假设原始维度为d，样本数目为m，因此特征矩阵X的维度为d×m，W的维度为d×d'。降维的时候，transpose(W)*X得到 d'×m的矩阵，它的每一列，即为降维后的向量。

2.数学推导

不过西瓜书里没有详细推导（我查了好几本书，包括DeepLearning和Hands On ML，都没有写详细推导，DeepLearning是作为一个练习让读者自己做）——为什么选取最大的d'个特征值所对应的特征向量就可以组成投影矩阵？结合DeepLearning里的部分过程，我补全了一个类似的证明。不排除有不对的地方，供有需要的朋友参考。

而 $d^Tx^{(i)}$ 为标量，转置与自身相等，上式进行简化

$$d^* = argmin_d \sum_i ||(x^{(i)} - x^{(i)T}dd||_2^2 \;\; subject\ to\ ||d||_2 = 1$$

再将每一个输入点叠加起来，我们得到

$$d^* = argmin_d \sum_i ||X - X^Tdd||_F^2 \;\; subject\ to\ d^Td = 1$$

Frobenius范数简化成（考虑约束条件 $d^Td = 1$）

$$\begin{aligned} &argmin_d \sum_i ||X - X^Tdd||_F^2 \\ &= argmin_d Tr((X - X^Tdd)^T(X - X^Tdd)) \\ &= argmin_d Tr(X^TX - X^TXd^T - dd^TX^TX + dd^TX^TXdd^T) \\ &= argmin_d - Tr(X^TXd^T) + Tr(dd^TX^TX) + Tr(dd^TX^TXdd^T) \\ &= argmin_d - 2Tr(X^TXdd^T) + Tr(dd^TX^TXdd^T) \\ &= argmin_d - 2Tr(X^TXdd^T) + Tr(X^TXdd^Tdd^T) \\ &= argmin_d - Tr(X^TXdd^T) \\ &= argmax_d Tr(X^TXdd^T) \\ &= argmax_d Tr(d^TX^TXd) \;\; subject\ to\ d^Td = 1 \end{aligned}$$

最后的优化目标可以利用 $\frac{\partial Tr(ABA^TC)}{\partial A} = CAB + C^TAB^T$ 以及拉格朗日乘数法来求解，可得最优的 d 是 X^TX 的最大特征值对应的特征向量。

上面的推导特定于 $l = 1$ 的情况，仅有一个主成分。一般来说，矩阵 D 由 X^TX 的前 l 个最大的特征值对应的特征向量组成（利用归纳法，将 D_{l+1} 表示为 D_l 的函数即可，需要两个辅助矩阵：单位对角矩阵 $R^{(l+1)\times l}$ 以及 $(0, 0\cdots, 0, 1)^T \in R^{l+1}$，省去证明过程）。

$$W^* = \underset{W^TW=I_d}{\operatorname{argmin}} Trace(-X^TWW^TX)$$

$$W^* = \underset{W^TW=I_d}{\operatorname{argmax}} Trace(X^TWW^TX)$$

由于 Trace(A*B)=Trace(B*A),上式转化为：

$$W^* = \underset{W^TW=I_d}{\operatorname{argmax}} Trace(W^TXX^TW)$$

由于 XX^T 为 d×d 实对称矩阵（本例中还至少是半正定矩阵），因此其存在 d 个实数（非负，因为半正定）特征值，设为 $\{\lambda_1, \lambda_2, ..., \lambda_d\}$（假设按从大到小排列），和 d 个正交的特征向量（不妨设为标准正交向量），设为 $\{v_1, v_2, ..., v_d\}$。亦即存在标准正交阵 $P = [v_1, v_2, ..., v_d]$ 满足 $P^TXX^TP = \Lambda <==> XX^T = P\Lambda P^T$，其中 $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_d)$ 为特征值组成的对角矩阵。

记 $W = [w_1, w_2, ..., w_d]$，则有

$$\begin{aligned} W^* &= \underset{W^TW=I_d}{\operatorname{argmax}} (w_1^TXX^Tw_1 + w_2^TXX^Tw_2 + ... + w_d^TXX^Tw_d) \\ &\underset{W^TW=I_d}{\operatorname{argmax}} (w_1^TXX^Tw_1 + w_2^TXX^Tw_2 + ... + w_d^TXX^Tw_d) \\ &\leq \underset{w_1^Tw_1=1}{\operatorname{argmax}} (w_1^TXX^Tw_1) + \underset{W^TW=I_d}{\operatorname{argmax}} (w_2^TXX^Tw_2 + ... + w_d^TXX^Tw_d) \end{aligned}$$

由 Rayleigh 熵最大最小定理（自己证明也很简单），上述不等式左侧为 $\lambda_{\max}(XX^T) = \lambda_1$。

根据 $W^TW = I_d$ 可知上述不等式右侧的 $w_2 \in \{span(v_2, v_3, ..., v_d) \cap (w_2^Tw_2 = 1)\}$。

因为用标准正交基可以表示任何向量，于是对任意的 w_2 可以用坐标线性表示为，

(a) PCA的目标

d维空间的向量x映射到d'维空间中y（PCA考虑的是用线性映射），同时有一个重构映射（解码过程，y是x的编码），设g(y)=W*y，为了简化模型，PCA限定W是列单位正交的，PCA希望解码之后的向量和原向量接近，即

$$y^* = \arg\min_y \| x - g(y) \|_2$$

对等式右端平方之后（不影响最小化问题），得到

$$\begin{aligned}
&\arg\min_y [(x-g(y))^T (x-g(y))] \\
&= \arg\min_y (x^T x - g(y)^T x - x^T g(y) + g(y)^T g(y)) \\
&= \arg\min_y (-2x^T g(y) + g(y)^T g(y)) \\
&= \arg\min_y (-2x^T W y + y^T W^T W y)
\end{aligned}$$

对于上述最小化问题，可以利用偏导数等于0得到g(y)，计算可得

$$y = W^T x$$

于是，对于原向量x，降维再还原得到的向量为：

$$WW^T x$$

对于整个特征表而言，类似的有降维再还原之后的矩阵为W*transpose(W)*X，PCA希望它和X越接近越好。用矩阵范数来衡量二者的接近程度，如用Frobenies范数来表示（矩阵每个元素的平方和再开方，不同的范数是等价的，用其他矩阵范数不影响结果）。
因此PCA的目标是求解W，使得：

$$W^* = \arg\min_{W^T W = I_{d'}} \| X - W * W^T * X \|_F$$

(b) 求解

$$W^* = \arg\min_{W^T W = I_{d'}} \| X - W * W^T * X \|_F$$

$$W^* = \arg\min_{W^T W = I_{d'}} Trace[(X - W * W^T * X)^T * (X - W * W^T * X)]$$

上式中Trace表示矩阵的迹（对角线的和），用到了性质 $\|A\|_F = Trace(A^T * A)$ 。

$$W^* = \arg\min_{W^T W = I_{d'}} Trace(X^T X - X^T W W^T X - X^T W W^T X + X^T W W^T W W^T X)$$

$$W_2 = [v_2, \ldots, v_d] * \begin{pmatrix} w_{22} \\ \ldots \\ w_{2d} \end{pmatrix},$$

因此

$$\begin{aligned}
&\arg\max_{W^T W = I_{d'}} (w_2^T X X^T w_2) = \arg\max_{W^T W = I_{d'}} \{ (w_{22}, \ldots, w_{2d}) * \begin{bmatrix} v_2^T \\ \ldots \\ v_d^T \end{bmatrix} * X X^T * [v_2, \ldots, v_d] * \begin{pmatrix} w_{22} \\ \ldots \\ w_{2d} \end{pmatrix} \} \\
&= \arg\max (\lambda_2 w_{22}^2 + \lambda_3 w_{23}^2 + \ldots + \lambda_d w_{2d}^2) \le \lambda_2
\end{aligned}$$

由于 $w_2 = v_2$ 时等号成立，于是有

$$\arg\max_{W^T W = I_{d'}} (w_2^T X X^T w_2) = \lambda_2 。$$

于是有

$$\begin{aligned}
&\arg\max_{W^T W = I_{d'}} (w_2^T X X^T w_2 + \ldots + w_{d'}^T X X^T w_{d'}) \\
&\le \lambda_2 + \arg\max_{W^T W = I_{d'}} (w_3^T X X^T w_3 + \ldots + w_{d'}^T X X^T w_{d'})
\end{aligned}$$

以此类推，可以得到：

$$\arg\max_{W^T W = I_{d'}} (w_1^T X X^T w_1 + w_2^T X X^T w_2 + \ldots + w_{d'}^T X X^T w_{d'}) \le \lambda_1 + \lambda_2 + \ldots + \lambda_{d'}$$

同时，容易验证，当 $w_1 = v_1, \ldots, w_{d'} = v_{d'}$ 时，

$$w_1^T X X^T w_1 + w_2^T X X^T w_2 + \ldots + w_{d'}^T X X^T w_{d'} = \lambda_1 + \lambda_2 + \ldots + \lambda_{d'}$$

综上，有

$$\arg\max_{W^T W = I_{d'}} (w_1^T X X^T w_1 + w_2^T X X^T w_2 + \ldots + w_{d'}^T X X^T w_{d'}) = \lambda_1 + \lambda_2 + \ldots + \lambda_{d'},$$

且等号成立的条件是 $w_1 = v_1, \ldots, w_{d'} = v_{d'}$，即W为 XX^T 的最大的d'个特征值对应的特征向量为列组成的矩阵。

证毕。

3.总结

（1）通过上述推导，感觉自己应该更清楚了PCA的原理：

PCA的目标是通过降维之后的向量，再还原回来的之后和原向量最接近（都是用线性映射），为了实现这个目标，得到的投影矩阵恰好是，原数据集的协方差矩阵的前d'个主成分为列组成的矩阵。

（2）从上述推导过程，也可以看到PCA和“不含非线性元素的自编码器”是在某些限定之下等价的。这在某些自编码器的资料里也常被提到，它们的思想都是相同的————通过线性编码和解码，使得输入和输出的差尽量小。

主成分分析降维原理——PCA数学推导

3、PCA

3.1 PCA 原理

假设 X={ x_1,...x_N }，$x_i \in \mathbb{R}^D$，PCA 的目标是找到一个低维空间能够表示数据并尽可能的与 X 相似。

再了解PCA之前，先了解一些概念：

a D 维空间的向量可以用基向量（basis vectors）的线性组合表示：$x_n = \sum_{i=1}^{D} \beta_{in} b_i$，这里的基向量是指D维空间中的向量，类似坐标轴。

b 由 **a** 可知：$\beta_{in} = x_n^T b_i$

c 从 D 维空间中选择前 M 个 b_i 作为正交基础向量（orthogonality basis vectors），即 $B = [b_1|\dots|b_M]$

向量 x 的投影向量为：$\tilde{x} = BB^T x$，$B^T x$ 叫做 coord or code。

PCA的主要想法是找到一个低维，可以用更少的基向量组成的 $\tilde{x}_n$ 表示 x_n，假设在M维空间中数据是中心化的，即数据的均值为0（$E(x) = 0$）。假设 $b_1 \sim b_D$ 是 $\mathbb{R}^D$ 的正交基础向量，则：

向量 x_n 在 M维空间的投影为：$\tilde{x}_n = \sum_{i=1}^{M} \beta_{in} b_i + \sum_{i=M+1}^{D} \beta_{in} b_i, \tilde{x}_n \in \mathbb{R}^D$

在PCA中会忽视第二项，所以 $\tilde{x}_n = \sum_{i=1}^{M} \beta_{in} b_i, \tilde{x}_n \in \mathbb{R}^D$

基向量 $b_1 \sim b_D$ 构成的子空间叫做 principle subspace, 即 $b_1 \sim b_D$ span the principle subspace.

虽然 $\tilde{x}_n$ 是 D 维向量，但是存在于 D 维的子空间 M 维中。

3.2 PCA 运用注意点

PCA算法中需要计算数据的协方差矩阵（数据的协方差矩阵是不稳定，因此需要中心化（中心化不等于归一化））以及对数据进行中心化处理（均值为0（新的数据值=数据值-均值））。中心化并不影响数据之间的相关性。

D维空间的协方差矩阵是 $D \times D$ 的矩阵，若D是高维的，则计算协方差矩阵的特征值和特征向量是昂贵的，因此可以对协方差矩阵进行处理。

假设数据点是少于纬度的，即数据点为 $x_1, \dots, x_N \in \mathbb{R}^D$，$X = \begin{bmatrix} x_1^T \\ \dots \\ x_N^T \end{bmatrix} \in \mathbb{R}^{N \times D}$，其中N<D。

$S = \frac{1}{N} X^T X$，S 的秩为N，即 $rank(S) = N$，所以有D-N+1个值为0特征值，即矩阵不是满秩（矩阵的行列之间是线性相关的（linearly dependent）即矩阵中有一些重复（redundancies））。

接下来让 $D \times D$ 的协方差矩阵变成满秩的，即矩阵中不存在为 0 的特征值。

$$
\begin{aligned}
Sb_i &= \lambda_i b_i, i = M+1, \dots, D \\
\frac{1}{N} X^T X b_i &= \lambda_i b_i \\
\frac{1}{N} X X^T X b_i &= \lambda_i X b_i (\text{两边同乘一个} X) \\
\frac{1}{N} X X^T c_i &= \lambda_i c_i (X b_i = c_i, \frac{1}{N} X X^T \in \mathbb{R}^{N \times N}) \\
\frac{1}{N} X^T X X^T c_i &= \lambda_i X^T c_i (\text{两边同乘一个} X^T) \\
S X^T c_i &= \lambda_i X^T c_i
\end{aligned}
$$

$S = \frac{1}{N} X^T N \in \mathbb{R}^{N \times N}$，计算 的协方差矩阵要比计算 $D \times D$ 的代价小。

上述中需要调节的参数有 β_{in}, b_i：，对此可以构建平方差损失函数求得：

$$J = \frac{1}{N} \sum_{n=1}^{N} \|x_n, \tilde{x}_n\|^2$$

找到最优的基向量，对 β_{in}, b_i 求偏导数，令其偏导数为 0，即可求得。

除了令偏导数为 0 进行参数优化，还可以用 chain rule 进行优化：

$$
\begin{aligned}
\frac{\partial J}{\partial \{\beta_{in}, b_i\}} &= \frac{\partial J}{\partial \tilde{x}_n} \times \frac{\partial \tilde{x}_n}{\partial \{\beta_{in}, b_i\}} \\
\Rightarrow \frac{\partial J}{\partial \tilde{x}_n} &= -\frac{2}{N}(x_n - \tilde{x}_n)^T \\
\Rightarrow \frac{\partial \tilde{x}_n}{\partial \beta_{in}} &= b_i, i = 1, \dots, M \\
\frac{\partial J}{\partial \beta_{in}} = \frac{\partial J}{\partial \tilde{x}_n} \times \frac{\partial \tilde{x}_n}{\partial \beta_{in}} &= -\frac{2}{N}(x_n - \tilde{x}_n)^T b_i \\
&= -\frac{2}{N}(x_n - \sum_{i=1}^{M} \beta_{in} b_i)^T b_i \\
&= -\frac{2}{N}(x_n^T b_i - \beta_{in} b_i^T b_i), b_i^T b_i = 1 \\
&= -\frac{2}{N}(x_n^T b_i - \beta_{in}) = 0
\end{aligned}
$$

下

$$
\begin{aligned}
\tilde{x}_n &= \sum_{i=1}^{M} \beta_{in} b_i \\
&= \sum_{i=1}^{M} (x_n^T b_i) b_i \\
&= \sum_{i=1}^{M} (x_n^T b_i) b_i \\
&= (\sum_{i=1}^{M} b_i b_i^T) x_n
\end{aligned}
$$

$\sum_{i=1}^{M} b_i b_i^T$ 是投影矩阵。

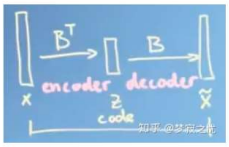
$$
\begin{aligned}
x_n &= \sum_{i=1}^{M} \beta_{in} b_i + \sum_{i=M+1}^{D} \beta_{in} b_i \\
&= (\sum_{i=1}^{M} b_i b_i^T) x_n + (\sum_{i=M+1}^{D} b_i b_i^T) x_n
\end{aligned}
$$

$\tilde{x}_n$ 与 x_n 相比少了第二项，所以 $\tilde{x}_n$ 是 x_n 的近似值（approximation）。

原始向量减去投影向量，即：

$$
\begin{aligned}
x_n - \tilde{x}_n &= (\sum_{i=M+1}^{D} b_i b_i^T) x_n \\
&= \sum_{i=M+1}^{D} (b_i^T x_n) b_i
\end{aligned}
$$

3.3 PCA 的其它解释



如上图所示，高维向量 X 通过基向量（B^T）投影到低维空间 Z（$= B^T X$）。B矩阵的列是数据协方差矩阵的特征向量，这些特征向量对应着大的特征值，Z 值是数据点相对于基向量的坐标，这些坐标对应着主要子空间，Z 也叫数据点的 code。低维的也可以转向高维，通过 BZ 即可将 Z转换到原来的数据空间。

通过构建平方差函数来找到PCA的最优参数，可将PCA看做是线性自动编码（linear auto encoder）。自动编码（autoencoder）对数据点X进行编码（encode）然后试图将其解码（decode）成一个类似X的数据点。将数据点映射到code叫做 encoder。将 code 映射到原始数据空间叫做 decoder。

若 encoder 和 decoder是线性映射，则PCA参数优化方法是用平方差损失函数；若将线性映射（linear mapping）换成非线性映射 (non-linear mapping)，则得到 non-linear autoencoder，如在深度自动编码器中，用深度网络代替编码器和解码器。自动编码 器的损失函数叫做 encoding loss。

PCA的另一个解释与信息论有关。可以将code看做是原数据点被压缩的版本，当用code重构原数据点时，不会得到与原始数据点完全一样的数据点，会得到一个稍微失真或有噪声的版本，意味着压缩真观上是有损的。压缩是通过最大化原始数据点与压缩的code之间的相关性，这跟 mutual information 相关。类似PCA损失函数的最优化，这里是最大化 mutual information。mutual information是信息论里一个重要的概念。在重构损失函数中，PCA是最小化数据在正交互补空间的方差，等价于最大化数据在主要子空间的方差。在信息论中最大化 mutual information 就是最大化方差，可以解释为尽可能的保留信息。

从隐藏变量模型（latent vatiable model）的角度看，Z是一个未知的低维空间，这个未知空间与X存在线性关系，因此X可以写成 $X = BZ + \mu + \epsilon$：，假设噪声 ϵ 是各向同性的（isotropic），则 $\epsilon \sim N(0, \sigma^2 \mathrm{I})$。进一步假设Z的分布式标准正态分布，即 $p(Z) = N(0, \mathrm{I})$。由上可以写出模型的似然函数：

$$
\begin{aligned}
p(X|Z) &= N(X|BZ + \mu, \sigma^2 \mathrm{I}) \\
p(X) &= \int p(X|Z) p(Z) dZ \\
&= N(X|\mu, BB^T + \sigma^2 \mathrm{I})
\end{aligned}
$$

模型的参数有 B, μ, ϵ，可以用最大似然估计求参数，发现 μ 是数据点的均值，B是包含与最大特征值相关的特征向量的矩阵。

$$p(Z|X) = \frac{p(X|Z) p(Z)}{p(X)}$$

为得到数据在低维空间的code，可以用贝叶斯理论，将Z和X之间的线性关系进行转换。

从上式可以看出两向量之差位于我们忽视的子空间里，即不包含主要子空间，该子空间是主要子空间的正交互补空间。

由上，损失函数可以重新定义为：

$$
\begin{aligned}
J &= \frac{1}{N}\sum_{n=1}^{N}\|x_n, \tilde{x}_n\|^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}\left\|\sum_{i=M+1}^{D}(b_i^T x_n)b_i\right\|^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}(b_i^T x_n)^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D} b_i^T x_n x_n^T b_i \\
&= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D} b_i^T x_n x_n^T b_i \\
&= \sum_{i=M+1}^{D} b_i^T(\frac{1}{N}\sum_{n=1}^{N} x_n x_n^T)b_i, S = \frac{1}{N}\sum_{n=1}^{N} x_n x_n^T, S是协方差矩阵 \\
&= \sum_{i=M+1}^{D} b_i^T S b_i \\
&= trace((\sum_{i=M+1}^{D} b_i b_i^T)S)
\end{aligned}
$$

$\sum_{i=M+1}^{D} b_i b_i^T$ 是投影矩阵（projection matrix），这个投影矩阵将数据协方差投影到主要子空间（principle subspace）的互补空间中，意味着重构的损失函数是将数据的方差投影到我们忽视的子空间去，因此最小化损失函数等价于最小化数据在忽视的空间的方差。

若 b_1 是 principle subspace， b_2 是 orthogonal complement（the space we ignore），则：

$$b_i^T b_j = \delta_{ij}; i = j, \delta_{ij} = 1; i \neq j, \delta_{ij} = 0$$

最小化(S)：$J = \sum_{i=M+1}^{D} b_i^T S b_i = b_2^T S b_2$

约束：$b_2^T b_2 = 1$

根据**拉格朗日乘子[1]**得：

$$
\begin{aligned}
L &= b_2^T S b_2 + \lambda(1 - b_2^T b_2) \\
\Rightarrow \frac{\partial L}{\partial \lambda} &= 1 - b_2^T b_2 = 0 \Leftrightarrow b_2^T b_2 = 1 \\
\Rightarrow \frac{\partial L}{\partial b_2} &= 2b_2^T S - 2\lambda b_2^T = 0 \Leftrightarrow S b_2 = \lambda b_2
\end{aligned}
$$

因此：$J = b_2^T S b_2 = b_2^T b_2 \lambda = \lambda$（$\lambda$ 是数据的协方差矩阵的特征值）。

当 λ 是最小特征值时，平方损失函数是最小的，意味着 b_2 是我们忽视的空间的对应的特征向量， b_1 是主要子空间是数据协方差矩阵的最大特征值的特征向量。 b_1, b_2 之间是正交的，因为协方差矩阵是对称的。

损失函数的一般形式：

$$
\begin{aligned}
S b_i &= \lambda_i b_i, i = M+1, .., D \\
J &= \sum_{i=M+1}^{D} \lambda_i
\end{aligned}
$$

当取的基向量（base vector）是被忽视子空间的数据协方差的最小特征值的特征向量时，损失函数是最小化； 这相当于主要子空间是由数据协方差矩阵的M个最大特征值的特征向量组成。大的特征值对应大的方差，方差方向由特征向量给出。

主要子空间是标准正交的（orthonormal ），协方差矩阵的特征向量与大的特征值之间是相关的。

W14 - Clustering

14_100. Unsupervised learning introduction __Clustering

Training example: unlabeled.

1. Compare supevised learning and unsupervised learning

>1. Supervised learning:
Supervised learning problem: given a set of labels to fit a hypothesis to it.

Traing set : is labeld;
Goal: found decison boundary to seperate postivie or negative examples.

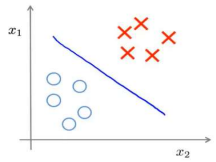
>2. unsupervised learning:
Unupervised learning problem: given a set of unlabels, try to find some structure in the data.

Traing set : do not have any labels associated with it. only {x1, x2, ..xm} , no labeled y.

E.g: one structure algorithm may find in right pic. that this data set has points group into two seperate clusters.

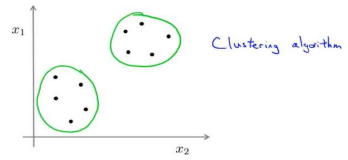
Clustering algorthm: find structure in (unlabeled) data sets that could group into different seperate clusters.

Supervised learning



Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots, (x^{(m)}, y^{(m)})\}$

Unsupervised learning



Training set: $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$

2. Clustering algorithm:

Clustering algorthm: find structure in data sets that could group into different seperate clusters

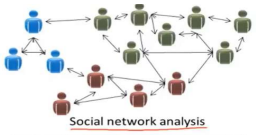
- Organize large computer clusters: try to find out which machines tend to work together (if put them work together, can make data center work more efficiently)

- Social network analysis: Given knowledge about which friends you email/facebook / circle friends the most, automatically classify which are cohesive groups of friends, which are groups of people that all know each other.

-Market segmentation: have huge databases of cutomer informaton, and automatically discover market segments and automatically group customers into different market segments. (do not know in advance market segments and customer group)

-Astronomical data analysis: clustering algorithms give usefuel theory of how galaxies are formed.

Organize computing clusters



Social network analysis

Market segmentation

Astronomical data analysis

14_101. K-means algorithm__Clustering

K-means algorithm: by far the most popular, widelly used clustering algorithm.

1. K-means algorithm

Input:
>1. K: number of cluster (want to find in data);

>2. Training set {x1, x2, ...xm} (unlabeled)
Note: x (i) is N dimensional, rather than n+1 : drop x0=1.

K-means algorithm

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

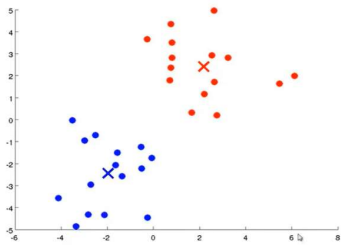
$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

2. Process:

>1. Randomly initailize K cluster centroids u1, u2, ..uK (n dimensional vector)

> 2. Cluster assignment step:
Found the cluster centroid that is closest to example x(i), and set x(i) as member of this cluster.

c(i): index of cluster centroid from 1 to K, that is closest to x(i)
for each example x(i), find out which cluster is closest to it: using distance -- min||x(i) - u_k'|| over k, ----> k optima (cluster)--->set c(i) =k for x(i)
Note: c(i): i is from 1 -K.



>3. Move cluster centroids:
move each cluster centroids position to the average of above training examples that have set to this cluster in step 2.

u_k := average (mean) of poins assigned to cluster k.

Repeate step 2 & step 3 till cluster classificaiton converge.
(note: how judge converged: cluster centroids : will not updated anymore in step3?
u_k==constant : k=1, 2, ..K)

Note: if no example assigned to a cluster u_k', then more common is just eliminate that cluster centroid.
or if do want K number cluster , then reinitialize clustertroids.

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$
Repeat {
Cluster assignment step: for $i = 1$ to m
$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$
Move centroid: for $k = 1$ to K
μ_k := average (mean) of points assigned to cluster k
}

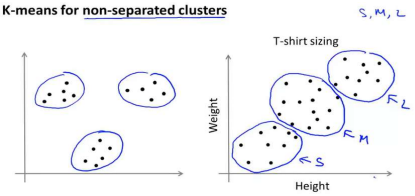
3. K-means for non-separated clusters

for data set that is pretty well sperated , easy for K-means to do clustering.
For date non-separted -not be several well separted cluster: K-means also works well:

e.g: right pic: training data is peope height and weigth, not well separated.
--->based on this to cluster into 3 size group (S, M, L) for T-shirt sizing

run K-mean algorithm, will seperated data into the 3 groups.
kind of like Market segmentation.

K-means for non-separated clusters



14_102. Optimization objective__Clustering

0: optimizing object Purpose

Both supervised algorithm and unsuspervised algorithm have object/cost function to optimize.
For clustering, optimizing objectiv helps for two purpose :
1. degugging algorithm and make sure algorithm is running correctly.
2. Find find better cluster and avoid local optima

1. K-means optimization objective:

>1. Paramters

>>1. c(i) = index of cluster (1, 2, ...K) to whic example x(i) is currently assigned;

>>2. u_k = cluster centroid: which is the locaton of cluster centroid k.
e.g: x(i)-->cluster 5, c(i)=5, u_c(i)= u_5

>>3. u_c(i) = cluster centroid of cluster to which example x(i) has been assigned.

K-means optimization objective

$c^{(i)}$ = index of cluster (1,2,...,K) to which example $x^{(i)}$ is currently assigned
μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)
$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

Optimization objective:

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

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

>2. Optimization objective /cost function / distortion cost function:

J(c(1), c(2), ...c(m), u_1, u2, ...u_K): = 1/m sum over all exmaple: ||x(i) - u_c(i) ||^2.

>>1. paramter c(1) to c(m), u1 - u_K is varing as the algorithm runs.

>>2. average of squared distance of: each example to the cluster centroids it is assigned to

>>3. Object: min J: try to find c(1) - c(m), and u_1 - u_K to minimize J

>3. K-means algorithm: to cost function

>> 2. Cluster assignment step:
Found the cluster centroid that is closest to example x(i), and set x(i) as member of this cluster.

cluster assign step: is minimizing cost function with respect to paramters c(1), c(2), ...c(m), while holding cluster centroids u_1, ..u_k fixed. as is minizing the distance of example (i) to it's assigned cluster centroid.

>>3. Move cluster centroids:
move each cluster centroids position to the average of above training examples that have set to this cluster in step 2.

Move cluster centroids: choosing cluser centroids u_1 - u_K, so it minimizes the cost function J with respect to the location of cluster centroids.

(note: k-means process per se : is minimizing cost function J: data 's distance to it's cluster centroid)

Summary:
K-means is taking the two sets of varaible and partitioning them into two halves c(i) and u_i. and then first optimize J with respect to the variable c(i), then minimize J with respect to the variable u_k. and then keep on iterationing.

14_103. Random initialization__Clustering

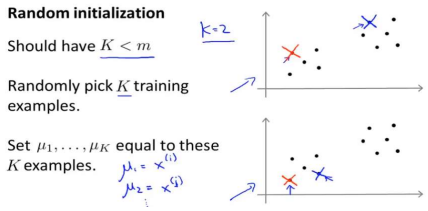
K-means fisrt step: Randomly initialize K cluster centroids u_1, u_2, ..u_K

1. Random initialization:

>1. Should have K number < m (training data size)

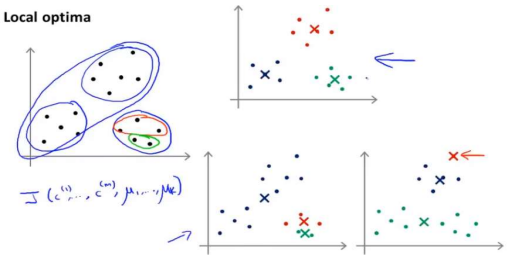
>2. Randomly pick K training examples:
Set u_1, u_2, ..u_K equal to these K training examples.

E.g: K=2, then randomly pick two examples as cluster centroids and run K-means



2. Random Intialzating problem:

K-means may converge to diffrent solutions ,may get stuck in local optima , depending on how the clusters were initialized.
And not doing a good job in minizing cost function J.



3. Random Intialzating -method:

-->Increase possiblity of K-means find the best possible clustering:
Try multiple , random initializations: Insteading of initializing K-means once, intialize K-means lots of times.

e.g : choose 100 number as K-means running times (randome initializing cluster centroids 100 times):
for each time:
>Randomly intialize K-means;
>Run K-means. Get c(1) - c(m), u_1 - u_K.
>Compute cost function J(c, u)

end up 100 ways of clustering the data -->pick clustering that gave lowest cost J.

Random initialization

For i = 1 to 100 {
  Randomly initialize K-means.
  Run K-means. Get $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$.
  Compute cost function (distortion)
  $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$
}

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

Summary:

When **K is small (2-10)**, then doing **multiple initializing can sometimes ensure finding a better local optima** , better clustering data.

When K is large, multiple initializing does not help much. Having higher chance that first initializing will give a pretty decent solution already. Mulitple intializing may give a slightly better solution but maybe not that much.

14_104. Choosing the number of cluster__Clustering

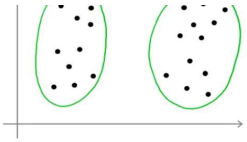
Most common way for choosing cluser number K:
choosing manuly, by looking at visualizations or by looking at the output of clustring algorithm or something else.

What is the right value of K?

1. Right value K:

Hard to choose K : as it is often ambious how many clusters there are in the data. And for unsupervise problem, there is not always a clear cut answers.

E.g: right pic: could be 4 cluster, or 2 cluster



2. Choosing the vlalue of K:

>1. Elbow method:

>>1. Ploting J with different K: run K-means (maybe multiple initializing or may not-only once) with different K, and compute distortion cost J.

>>2. Find in the above curve: 'elbow' point, then choose K corresponding to this elbom.
Distorting go down rapdily before elbow point, and after this point, distorttion goes down rather slowly.

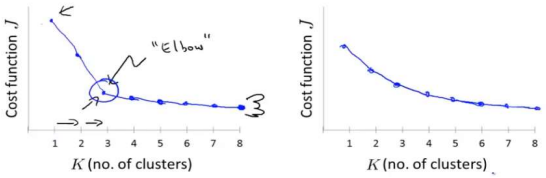
>2. Elbow method problem:
Elbow method do not use often, as most of time K-means will end up with a distortion curve-K, looks much more ambiguous: no clear elbow.
e.g : in right pic. distortion continuously goes down, K=3, 4,5 6, seems all good-no clear elbom.

if have clear elbow in distortion curve, ok to use Elbom method. but more often will end up smoother distorition curve-vs- K.

Elow worth for a shot , but do not expect it working for any particular problem.

Choosing the value of K

Elbow method:



>3. : choose K acc.to purpose:

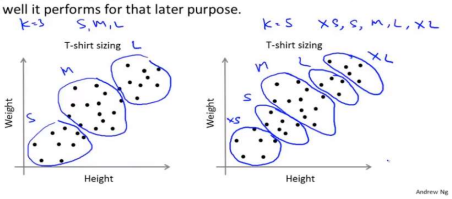
sometimes running K-means to get cluser to use for some later/downstream purpose.
Evaluate K-means based on a metric for how well it performs for that later purpose.

if downstream purpose gives an evaluation metric: then better way to determain number of clusters is to see how well different K clusters serve that later downstream purpose.

e.g: T-shirt sizing:
cluster value K is to use for t-shirt size design number.
Evalue K value / t-shirt size number on : how well T-shirt size fit customer: how happy customers will be.
More t-shirt size (K=5) fit cusmer better, while fewer size, could sell to customers more cheaply.-->**K value decided by commercial purpose**

Choosing the value of K
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

E.g.



Summary:
choosing K is still manuly. Elbom is one method but do not often use as K-means in fact has not clear elbow point.
Using later purpose for the data clustering to evalue how well different K value serve .

W15 - Dimensionality Reduction

15_107. Motivation I: Data compression__Dimensionality Reduction

Data compression: allow us to:
1. Compress the data and therefor use less computer memory or disk space.
2. Speed up learnig algorithms.

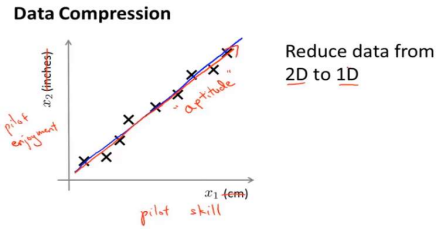
1. Data compression: 2D to 1D:

>1. Exmaples

e.g: right pic: x(i) have two feature x1, x2, : pretening length in differnt unit.
> these two feature is high relevent, -->high redundant representation.
>Redue data to one dimensional to just have one number/feature measuring the length.

when have large data, not hard to have highly redundant features like these.

e.g: 2: x(i) have two feature x1, x2. x1: pilot enjoyment; x2: pilot skill.
x1, x2 is highly relevent, reduce two feature x1, x2 to one 'aptitude' dimension.

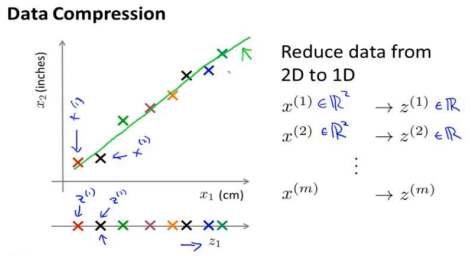


>2. Intuition:

Data compresion from 2D to 1D:
>1. Finding one dimension line/ new feature,
>2. project all 2D data (each training data , have two features) x(i) to that line;
>3. Measure the position of each data projection on the line/ new 1 dimension feature : one real number z(i).

Mapping:
x(i) belong to two dimension space, -------->Z(i) : one dimenson space (by projecting x(i))

after compresion, for each example now only need to keep around one real number.
>1. half memory requirement for how to store data.
>2. Algorithm run more efficiently.

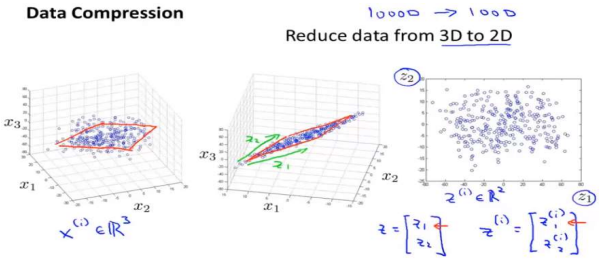


2. Data compression: 3D to 2D:
in reality may reduce feature space dimension from 10,000 to 1000. use 3D to 2D as example as this dimension could visulization.

>1. Trianing example x(i) belong to 3 dimnesion space (each training example have 3 features): x_1, x2, x3
>2. Project all training examples to a plane: z(i): two dimension space.
example z(i) now have two new features**: z_1, z_2.**

>3. Now could representing each example with two numbers

Mapping:
x(i): [x_1; x_2; x_3]T---------------------------> z(i) [z_1; z_2] T



15_108. Motivation II: Data Visualization__Dimensionality Reduction

Dimensionality reduction purpose:
1. Comress data;
2. Visualize data;

1. Data visualizaiton:

e.g:
> 1. right pic, have training examples with many features.

----> Try to undersatnd those data (with large feature space) better / visulize these data
(when have large features dimension, hard to plot training data to visulize)

>2. Using different feature to represent training data. right pic.reduce original many feaures to only new two features.
then plot these examples with new feature space.

Mapping x(i) belong to like 50 dimension sapce to 2D space
x(i) [x_1; x_2; ....x_50]T---------->z(i) [z_1; z_2]T

Note:
new feature z_1, z2 usually does not astride a physical meaning. It's often up to us to figure out roughly what these features means.

Data Visualization

Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Develop-ment Index	x_4 Life expectancy	x_5 Poverty Index (Gini as percentage)	x_6 Mean household income (thousands of US$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	...

[resources from en.wikipedia.org]

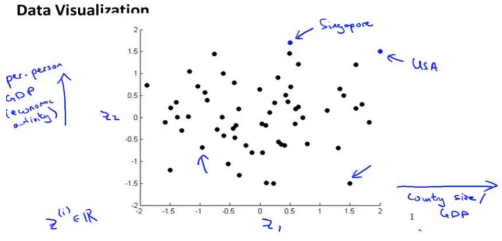
Data Visualization

Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
...	...	...

>3. Ploting example in new feature space:
by ploting examples with new features, may find new feature physical meaning.

E.g: right pic:
horizental feature z_1 may corresponding to country size / GDP;
Vertical feature z_2: may corresponds to GDP/ per captital.

new feature z_1, z_2 may can help to most succinctly capture really what are the two main dimensions of the variation amongst different countries.



Summary:
dimension reduction: above example reduce example featuer space from 50 to 2D. And then **by ploting examples in new feature space to understand data better.(Visulization)**

15_109. Principal component Analysis problem formulation__Dimensionality Reduction

PCA is by far the most common used method to do dimenson reduction.

1. PCA problem formulation:

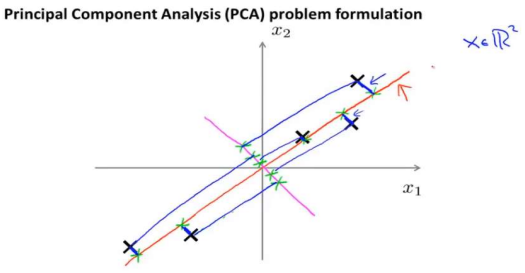
PCA function:
try to **find a lower-dimensional surface**, onto which to project the data, so as to **minimize all example's projection error.**

Projection error: distance of original example data to new projected postion on the lower dimensional surface

e.g: right pic:

if choose pink line instead of red line as new feature : projection error will be huge.
-->PCA method will choose red line instead of pink line as the new feature line.

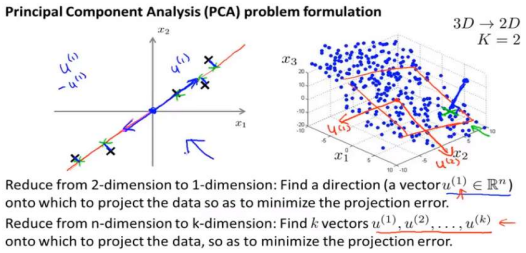
Note:
before appling PCA, it's standard practice to first perform mean normalization and feature scaling.
So feature x1, x2 should have zero mean, and should have comparable ranges of values.



2. PCA function:
>1. for dimension reducing from 2D to 1D:
Find a direciton (a vector u belong to n dimension space**: n is example original feature space/number.** Find a vector belong to 2 dimenson space in this case), onto which to project the data so as to minimize the projection error.

Note: for the u to project on, does not matter if give oppositive vector-u. the direction of vector is same.

>2. Reduce from n-dimension to k-dimenson:
PCA find k vectors u1, u2, uk, , onto which to project the data, so as to minimize the project error.
Project data on the linear subsapce spanned by this set of k vectors.



3. PCA is not linear regression

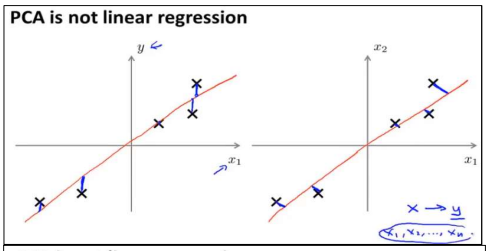
Linear regression:
>1. fitting a straight line to training data, so as to minimize the squared error of predictioin value (point on the line) to example (vertical distance in the ploting).
See left pic. Blue line;
>2. All the data set used to predict y.

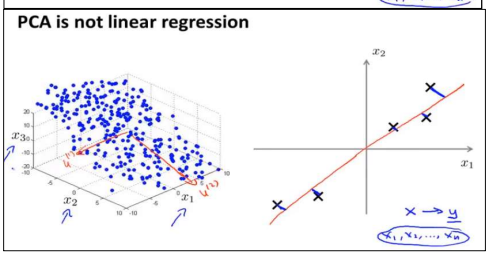
PCA:
>1. fitting a straight line to training data, so as to minize the distance of example projection (point on the line) to example
See right pic. blue line: shortest orthogonal distances （最短的正交距离）: shortest distance from example to the straight line

>2. No label y, only features x1, x2..xn, all the features are treated equaly (scaled before peforme PCA).
e.g: reduce from 3d to 2D:
try to find new feature directions u1, u2 onto which to project original data, and all of these original feauturs are treated symmetrically, no specail y trying to predict .

-->PCA is not linear regression even though at some cosmetic level , they might looks related , these are actually very different algorithms.

> this (minimize object differences) gives very different effects depending on training set.





15_110. Principal component Analysis problem formulation II_Dimensionality Reduction

1. Data preprocessing

Training set: unlabeled x(1), x(2), ..., x(m)

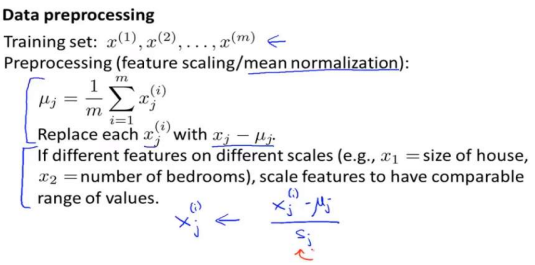
Preprocessing : (feature scaling / mean normalization)-same procedure with supevised learning:

>1. Mean normalization: x(i)- u -->training set have 0 average value.

>2. If different features on different scales, scale features to have comparable range of values.

(x(i) - u) _j/ s_j:

s_j: vitual values of feature j; could be max - min value, or more common use the standard deviation of feature j.

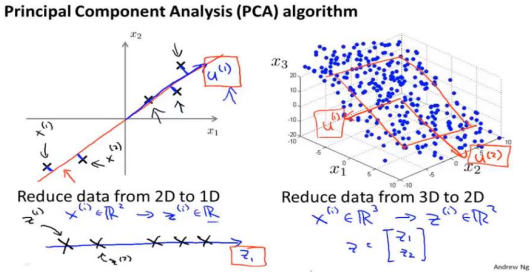


2. PCA algorithm

>1. PCA function:
try to find a lower-dimensional surface, onto which to project the data, so as to minimize all example's projection error.
(Note: PCA:
1. 选取方差大的方向作为主方向: project后， 样本间离散度大-信号多--等同于projection 后Position value big
note: 舍弃离散度小的projection 方向； 如噪声；
2. evaluation: 失真小;distance fractor: sum of all example distnace to it's projection point / all example distance sum-和linear regression object same??: minimize distance error to example and predictoin/projection point)
)

>>1: 2D to 1D: find a direction and project example to that line:
x(i): [x_1; x_2]T of 2D space ----->z(i): [z_1]: real value (1 D space): one number to specifiy the position of a point.

>>1: 3D to 2D:
x(i): [x_1; x_2; x_3]T of 3D space ----->z(i): [z_1; z_2]: 2 D space



3. PCA algorithm implementation procedure

Reduce data from n-dimensions to k-dimensions

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

>Compute 'convariance matrix'（协方差矩阵）Sigma: n x n
(note: 实对称矩阵:
对角线是方差-on condition performe 0 mean on training set;
other position value: is feature convariance matrix: representing independency of different features

2. projected feature space(映射后的降维空间里表达Y): convariance matrixY*YT: requirement
-->1. 对角矩阵: feature 不相关(!= 独立)
-->2. 样本离散度大 : ->specify : new space feature direction (& 失真度 & -->new feature 不相关??)
-->3.失真小: 样本与Projected point distance small
3. projected example Y: convariance matrix: A=Y* YT: =UT*X *XT*U = UT*U*sqrt(S)*Sqrt(S)*UT*U=S
-new feature y_i, 方差D(y_i)= E([y_i-E(y_i)]^2)=E(y_i^2)
D(y_i) = (Y*YT) _ii = S_ii

4. training set square length ||X||^2=trace(XT*X) = sum (s_ii)
5. projection loss = ||x(i)-p(i)||^2 = sum of (s__ii : for space not projected in .)

)

>Compute ' eigenvectors' 特征向量 of convariane marix
use right code in Octave: [U, S, V] = svd (Sigma)

svd: singular value decomposition （奇异值分解）;
Note:
>1. 'eig' function and 'svd' will give same matrix- as convariance matrix always satisfies a mathematical property: symmetric positive definite （正定矩阵）。
only svd is more numerically stable
>2. If use other language rather than octave/ matlab, -->find the numberical linear algebra library （数值线性代数库）that can compute svd.

Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

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

Sigma

Compute "eigenvectors" of matrix Σ: → Singular value decomposition

eig (Sigma)

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

$n \times n$ matrix.

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

$u^{(1)}, \ldots, u^{(k)}$

k

4. PCA computation result: from [U, S, V] = svd (Sigma) get

>>1. U = [u1, u2, ...u_n]; n x n metrics. u_i is the i_th directon/vector of new feature space, which example gonna to project on.
if reduce from n dimenson to k- dimenson, then just take the first k verctors: u1, u2, ..u_k;

U_reduce = [u1, u2, ..u_k] : n x k ; this metrix is used to reduce data dimension:
Z (k, 1) _i = U^T (k, n) * X (n,1)_i
x_i could be from training set ,dev set or test set.
(note:
example x(i) project on vector u1T= u1T*x(i)/ |u1| , (|u1| =1
UT=[u1T; u2T; ...ukT]
z(i) = UT * x(i) = [u1T*x(i); u2T; ...ukT * x(i)]

actually is preseting x(i) with less origian space vector:
x(i) = (a1, ...an)T * b_i (b_i: projection on n space vector: 基向量 a1, ..an)
y(i) = (a1, ...a_k, a_k+1, ..an)T*r_i(r_i: new position of x(i), represented by only k vector in n space

loss = x(i)-y(i) distance : to make loss small:
-->d_l / dr(i) =0--> r(i) = AT*x(i) -->
loss (i) = (a_k+1, ..an)T * r(i): 在为投影的方向/空间(原n space 的子空间: n-k+1空间)的损失

)

Principal Component Analysis (PCA) algorithm

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

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

k

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

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

$z \in \mathbb{R}^k$ $n \times k$ U_{reduce} $k \times n$ $n \times 1$

Andrew Ng

5. PCA algorithm summary:

>1. After mean normalization (ensure every feature has zero mean, **why?**) and optionally feaure scaling (do scaling if features is on very different ranges of values);
(-take each feature equally, not dominated by large number feature)

>2. Compute convariance matrix Sigma; = 1/m * X * X^T
X=[x(1), x(2), ...x(m)]: n x m metrix

>3. coding: [U, S, V] = svd (Sigma):
U_reduce = U (:, 1: k) # take first k vectors #
z = U_reduce^T *x : # example x presented by n features/n 个原空间的基向量 , change to k dimension repeseNttation z(其中k个原空间的基向量 #

Note: x(i) is n dimensonal, x (i)_0 =1 is not considered here.

Principal Component Analysis (PCA) algorithm summary

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

$$\text{Sigma} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T \quad X = \begin{bmatrix} — (x^{(1)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

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

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

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

15_111. Choosing the number of principal components__Dimensionality Reduction

1. Choosing K (number of principal components)

PCA object: minimize average squared project error;（向量距离）

Total variation in the data: average of emaple length in original feature space.

Principle of choosing : choosing the smallest value make below fraction <=0.01;
projection error / tatal variationi <=0.01 (1%)
another way interpresentation for above principle: choosing K so as "99% of variance is retained'

And so range of values from, you know, 90, 95, 99, maybe as low as 85% of the variables contained would be a fairly typical range in values. Maybe 95 to 99 is really the most common range of values that people use.

Choosing k (number of principal components)

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

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

Typically, choose k to be smallest value so that

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

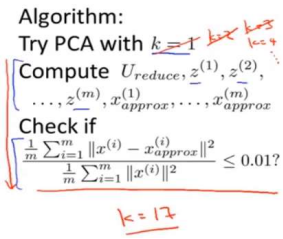
0.05 0.10

→ "99% of variance is retained"

95% 90%

For many data sets you'd be surprised, in order to retain 99% of the variance, you can often reduce the dimension of the data significantly and still retain most of the variance. Because for most real life data says many features are just highly correlated, and so it turns out to be possible to compress the data a lot and still retain you know 99% of the variance or 95% of the variance.

2. Choosing K algorithm:



> Method 1: Start by choosing K=1, and go through below:
>1. Run through PCA; --->compute all examples new postion in new feature space: z_1, z_2, ..z_m
>2. compute fractor: projection error / total validation. check if less than 0.01
>3. if ok, then use k=1. otherwise try next K=2, and run through above step1 & step2 again. till the choosen K meet fractor spec.

So how do you implement this? Well, here's one algorithm that you might use. You may start off, if you want to choose the value of k, we might start off with k equals 1. And then we run through PCA. You know, so we compute, you reduce, compute z1, z2, up to zm. Compute all of those x1 approx and so on up to xm approx and then we check if 99% of the variance is retained.

Then we're good and we use k equals 1. But if it isn't then what we'll do we'll next try K equals 2. And then we'll again run through this entire procedure and check, you know is this expression satisfied. Is this less than 0.01. And if not then we do this again. Let's try k equals 3, then try k equals 4, and so on until maybe we get up to k equals 17 and we find 99% of the data have is retained and then we use k equals 17, right?

That is one way to choose the smallest value of k, so that and 99% of the variance is retained.

---

above procedure seems horribly inefficient, we're trying k equals one, k equals two, we're doing all these calculations.

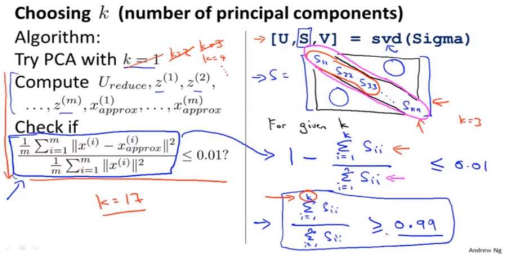
>Method 2: call svd function; [U, S, V] = svd (Sigma)

S: a square matrix (n, n) and diagonal matrix.

for a given k: fractor = 1- sum (s_ii) over K / sum sum (s_ii) over m

so we can then just slowly increase k, set k equals one, set k equals two, set k equals three and so on, and just test this quantity to see what is the smallest value of k that ensures that 99% of the variance is retained.

---> call the SVD function only once, and gives you S matrix ---> keep on doing this calculation by increasing the value of K in the numerator.
So this procedure is much more efficient, and this can allow you to select the value of K without needing to run PCA from scratch over and over.



Summaryize:

for choosing K, recommend when using PCA for compression is call SVD once in the covariance matrix, and then use this formula and pick the smallest value of K for which this fraction is satisfied.
(note bigger k, less loss as representing in a space more close to original space)

Summary:

A good way to explain the performance of your implementation of PCA , is actually to tell what was the percentage of variance retained: how well your hundred dimensional representation is approximating your original data set. Gives people a good intuitive sense of whether your implementation of PCA is finding a good approximation of your original data set.

Often find that PCA will be able to retain 99% of the variance or say, 95%, 99% , some high fraction of the variance, even while compressing the data by a very large factor. this is because high dimensional data sets, like a thousand dimensional data, very often tend to have highly correlated features, this is just a property of most of the data se**ts.-mathmetical proof?**

15_112. Reconstruction from compressed representation_Dimensionality Reduction

PCA has been used as a compression algorithm for reducing high(like 1,000)-dimensional data to lower space (like 100-dimensional) feature vector

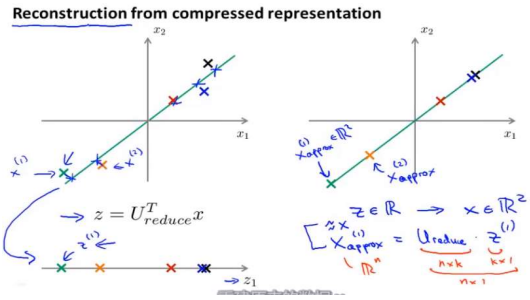
1. Reconstruction from compressed representation

e.g:
>1. Compressing: 2D to 1D : z(i) = U_reduce ^{T} * x(i)
x(i) project on one dimensional surface, then use a real number z1 to specify the location of these points after they've been projected onto this one dimensional surface.

>2. Restructure: 1D----->2D: x(i)_approx = U_reduce * z(i)
Objective: Given the point z1, go back to this original two dimensional space.
or In particular, given the point z (a real value), map this back to some approximate representation x (point in 2 dimension space).

>> go in the opposite direction of formula: x(i)_approx (n,1) **= U_reduce** (n, k) * **z(i)** (k, 1)

The intent of PCA, is if the square projection error is not too big, is that this x(i)_approx will be close to whatever was the original value x (i) that have used to derive z in the first place.
(note: svd
svd: X [x1, x2, ..xm] = U* M*V
-->Z=UT*X)
-->X=U*Z



note:
actually is preseting x(i) with less origian space vector:
x(i) = (a1, ...an)T * b_i (b_i: projection on n space vector：基向量 a1, ..an)
y(i) = (a1, ...a_k)T*r_i(r_i: new position of x(i), represented by only k vector in n space)
y_i: x(i) project point in n original space: use less 基向量 a1, ..a_k, position is these vector is r_(i): kx1
r_(i) ： is projection point position in a1, ..a_k space: kx1 (z(i)

loss = x(i)-y(i) distance : to make loss small:
-->d_l / dr(i) =0--> r(i) = A_kT*x(i) -->
loss (i) = (a_k+1, ..an)T * r(i): 在为投影的方向/空间(原n space 的子空间： n-k+1空间)的损失

Note:
>1. x(i)_approx calculated is the projection point of original data on lower dimension surface, represented in n dimension space. while z(i) is the projection points' representation in lower dimension space.
U_reduce here is for feature space transition, while point itself is same: projection point on lower space surface. U^T use for change from high dimension to lower dimension space (compressiong), U is for otherwise.

Above is a pretty decent approximation to the original data. So that's how you go back from your low dimensional representation z, back to an uncompressed representation of the data. We get back an approximation to your original data x. And we also call this process reconstruction of the original data where we think of trying to reconstruct the original value of x from the compressed representation.

15_113. Advice for appling PCA_Dimensionality Reduction

PCA can be sometimes used to speed up the running time of a learning algorithm.

1. Supervised learning speedup

Training example: x(1), y(1) ....x(m) y(m).

>1. large feature dimension problem:
Compute version problem: x(i) is high dimensional: image if 100x 100-->x(i) dimension is 10,000.
-->
if feed 10,000 dimensional feature vectors into logistic regression, or a new network, or support vector machine or what have you, just because that's a lot of data, that's 10,000 numbers,
it can make your learning algorithm run more slowly.make learning algorithm run more slowly.

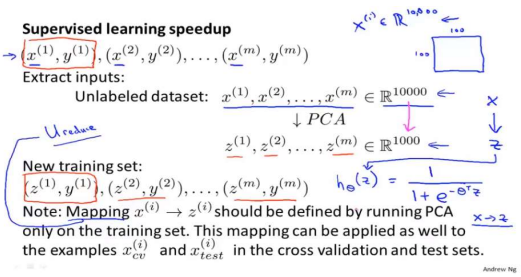
>2. Compressing dimenson process:
preprocessing: normalize and scallig feature of traiing set.
>>1. Extract inputs: **unlabeled data** set: x(1), ...x(m)

>>> First check our labeled training set and extract just the inputs
>>>Extract the X's and temporarily put aside the Y's. So this will now give us an unlabelled training set x1 .... xm.
>>> So just extract the input vectors as unlabeled data set.

>>2. Apply PCA on unlabeld datat set x(1), ..x(m) --->lower dimension data set z(1), ...z(m);

>>3. New training set: (z(1), y(1)), (z(2), y(2)), ... (z(3), y(3))

then feed new training set to algorithm, and try to make prediction.



Note:
1. when have new examples to predict , first map it throup same mapping training set have done to get the mapping point z and then feed z to algorithm
2. Mapping x(i) -----z(i) should be defined by runing PCA only on the training set.
then could apply the same mapping to other examples that in dev / test set.
this mapping does: feature mean normalization and scaled, computing U_reduce matrix-matrix parameters (k, u_i, z_i) should only fitting to training set.- found all the mapping paramters on the training set.

3. In reality we actually reduce the dimensional data 1/5, 1/10 and still retain most of the variance and barely effecting the performance.
in terms of classification accuracy, barely affecting the classification accuracy of the learning algorithm. And by working with lower dimensional data our learning algorithm can often run much much faster.

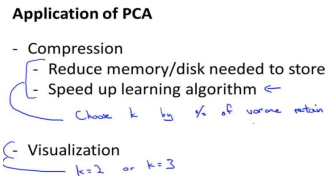
2. Application of PCA:

Main applications for PCA: compression and visulization

>1. Compression: < choose k acc. to variance retained

>> Reduce the memory or the disk space needed to store data
>> Speed up a learning algorithm.

In these applications, choosing K by the percentage of variance retained, and so for this learning algorithm, speed up application often will retain 99% of the variance. That would be a very typical choice for how to choose k.

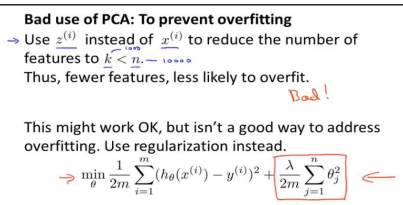


>2. Visualization: <k set to 2/3 for plotting
for visualization applications, we'll usually choose k equals 2 or k equals 3, because we can plot only 2D and 3D data sets.

3. Bad use of PCA: to prevent overfitting

> PCA for overfitiong- Bad PCA application
Use z(i) instead of x(i) to reduce the number of features to k < n.
Thus fewer features (algorithm parameter θ could set to 0), less likely to overfit.??
if k is 1,000 and n is 10,000, then if we have only 1,000 dimensional data, maybe we're less likely to over-fit than if we were using 10,000-dimensional

This might work OK, but is not a good way to address overfitting. Use regularization instead.



> Reason:

PCA process:
does not use the labels 'y', just looking at your inputs xi, and using that to find a lower-dimensional approximation to your data.
---> throws away some information / reduces the data dimension without knowing the values of y .

This is probably okay using PCA this way if 99 percent of the variance is retained.
if keeping most of the variance, but it might also throw away some valuable information.

And it turns out that if retaining 99% / 95%of the variance, it turns out that just using regularization will often give at least as good method for preventing over-fitting ,
and regularization will often just work better:

Regulization:
because when applying linear regression or logistic regression or some other method with regularization, this minimization problem actually knows what the values of y are, and so is less likely to throw away some valuable information. Whereas PCA doesn't make use of the labels and is more likely to throw away valuable information. -
PCA treat each original feature of training set x(i) equally by normalizinga and scaling-no concern labeled y, throw away some feature-->set some parameter θ to 0, while regulazation concerned labeled y in minimizing object (note: regularization penalty on each parameter/weight/feature is same, while penalty magnitude involed labled info.).

Summarize:
PCA good use to speed up your learning algorithm, but not to prevent over-fitting, which using regularization instead is really what many people would recommend doing.

4. PCA misuse in ML design system

PCA is often use to compress and visulizion.
but there is **pretty common misuse: consider PCA as standard process of ML design**

Often Common Design of ML system:
> Get a training set { ((x(1), y(1)), ((x(2), y(2)), ...((x(m), y(m) };
> Run PCA to reduce x(I) dimension to get z(i);
> Train logistic regression on { ((z(1), z(1)), ((z(2), z(2)), ...((z(m), z(m) };
> Test on test data: Map x(i)_dev/test to z(i)_dev / test. Run h(z)_θ on { ((z(1), z(1)), ((z(2), z(2)), ...((z(m), z(m) }.

-->problem: At the very start of a project, just write out a project plan like above with PCA inside.

Try Algorithm performance without PCA:
---> before you implement PCA, do whatever it is, take whatever it is you want to do and first consider doing it with your original raw data x(i), and only if that doesn't do what you want, then implement PCA before using Zi.

PCA is sometimes used where it shouldn't be

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

How about doing the whole thing without using PCA?

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

instead of putting PCA into the algorithm, just try doing whatever it is you're doing with the xi first. And only if you have a reason to believe that doesn't work, so that only if your learning algorithm ends up running too slowly, or only if the memory requirement or the disk space requirement is too large, so you want to compress your representation. but if only using the xi doesn't work, only if you have evidence or strong reason to believe that using the xi won't work, then implement PCA and consider using the compressed representation.

Because what I do see, is sometimes people start off with a project plan that incorporates PCA inside, and sometimes they, whatever they're doing will work just fine, even with out using PCA instead. So, just consider that as an alternative as well, before you go to spend a lot of time to get PCA in, figure out what k is and so on.

Summary:
1. PCA is an incredibly useful algorithm, when you use it for the appropriate applications:
> Use it mostly to speed up the running time of learning algorithms;
> Just as common an application of PCA, is to use it to compress data, to reduce the memory or disk space requirements, or to use it to visualize data.

2. PCA is one of the most commonly used and one of the most powerful unsupervised learning algorithms

W16 - Anomaly detection

16_115. Problem motivation__Anomaly detection

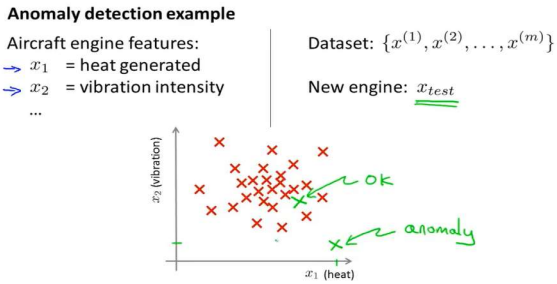
Anomaly detection:
A reasonably commonly use type of machine learning, and mainly for unsupervised problem, which in some aspects very similar to supervised learning problem.

1.Anomaly detection example:

aircraft engine: quality testing after production line.
>1. Measure features reflecting quailty for each product on production line:
x_1: heat generated;
x_2: vibration intensity...
...
x_n.

--->x(1) = [x_1; ..x_n]T;

>2. Measure m product, get unlabeled examples: x(1), ...x(m)
>3. Ploting unlabeled example x(1), ...x(m)
>4. Ploting new example x_test on above figure, to check if this point is anomal
---> quality issue/ need further check

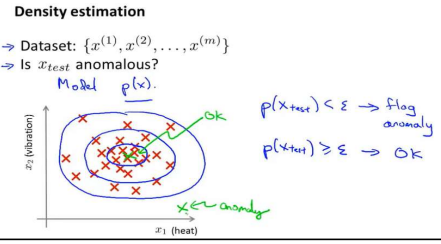


2. Density estimation:

for anomal detection:
> 1. **Training example**: x(1), ...x(m):
Usually assume training set are nomal or non-anomalous , unlabeled examples

>2. Build probability model: p(x): x is the features for examples.

>3. Algorithm to tell if new example is anomalous:
if p(x_test) <ε --> judge as anomalous example

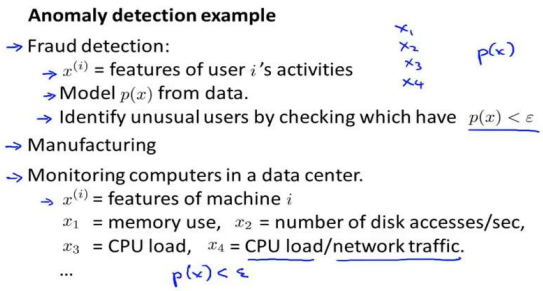


3. Anomaly detection example:

> Example 1: Fraud detection:
Most common application of anomaly detection is Fraud detection:
f you have many users, and if each of your users take different activities, y on your website or in the physical plant or something, you can
>>1. x(i)= features of user i's activities;
compute features of the different users activities: x_1 could be how often user log in; x_2 could be pages number visited, x_3 posts number user on the forum...

>>2. Build Model p(x) based on above data set
The probability of different users behaving different ways, / probability of a particular vector of features of a users behavior

>>3. Identify unusual users by checking which have p(x) < ε:
and maybe send the profiles of those users for further review, Or demand additional identification from those user



This sort of technique will tend to flag the users that are behaving unusually, not just users that maybe behaving fraudulently. So not just constantly having stolen or users that are trying to do funny things, or just find unusual users. But this is actually the technique that is used by many online websites that sell things, to try identify users behaving strangely that might be indicative of either fraudulent behavior or of computer accounts that have been stolen.

> Example 2: Monitoring computers in a data center:

Have a lot of machines in a computer cluster or in a data center, we can do things like
>>1. compute features at each machine.:
e.g: x_1: memory used, x_2: number of disc accesses, x_3: CPU load, x4: CPU load / network traffic

>>2. Model p(x):
Then given the dataset of how your computers in your data center usually behave, you can model the probability of x, so you can model the probability of these machines having different amounts of memory use or probability of these machines having different numbers of disc accesses or different CPU loads and so on.
And if you ever have a machine whose p(x) , is very small then you know that machine is behaving unusually

and this is actually being used today by various data centers to watch out for unusual things happening on their machines.

16_116. Gaussian distribution__Anomaly detection

Gaussian distribution also called the normal distribution

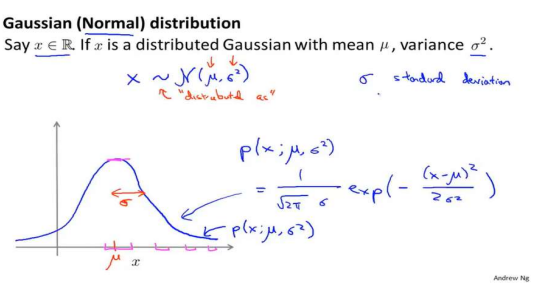
1. Gaussian (Normal) distribution

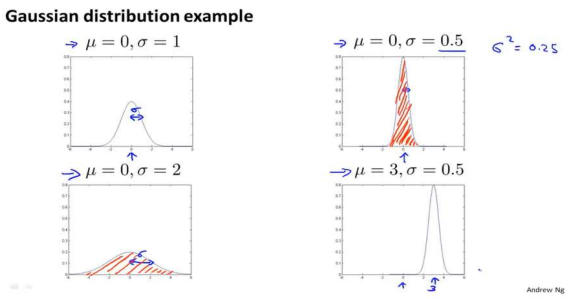
Probability distribution of X: X ~ N (u, σ^2):
controlled by two parameter:
u: mean of x;
6 : standard devitaion of x from u, and is the width of p(x) curve/Gaussian density

2. Gaussian (Normal) distribution example

u: control mean postion of gaussian density;
6: control width of gaussian density:
>Small: gaussian density thinner /smaller width and taller;
>Big: Gaussian density much fatter/ wider and shorter

Guassian distribution property: area under gaussian curve is =1: Sum of gaussin density over x width =1
-->wider in range, shorter in tall





3. Paramter estimation

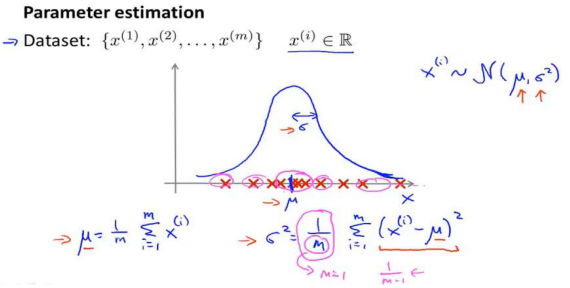
Paramter estimation problem:
e.g: dataset: x(1), ... x(m), below to one space R (real number)
Plot these examples on x axile. And Suspect each example distributed acc. to Gaussian distribution.

-->problem: do not know Guassian paramter: u, 6

Parmater estimation: estimate Guassian paramter: u, 6 r on training example:
u = average of example x(i);
6^2 = average of example variance from u.

Note: In statics （数理统计） - maximum likelihood estimation (极大似然估计),
1. Above estimation is the maximum likelihood estimation of u, 6

2. in statics, 6^2= sum of example variance / (m-1), but not m. but in practice no much difference to use m or m-1 as it's a large training set size.



16_117. Algorithm_Anomaly detection

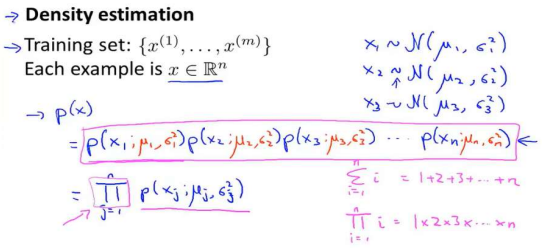
1. Density estimation

>Training set: unlabeled { x(1), ...x(m) }
each example is a feature vector in R^n--->training set is m feature vectors.

>Model p(x): muliply probability of each feature:
to figure out what are high/ lower probability features,
x (i): feature vector
p(x(i)) probability of first feature, second feature, 3rd feature... n_th feature.

p(x(i)) = p(x_1, u_1, 6_1^2) * p(x_2, u_2, 6_2^2) * .....p(x_m, u_m, 6_m^2)

assume:
1. Each feature distribution acc. to Gaussian density :x_i ~ N (u_i, 6_i^2)
2. in Statics Each feature is independent to other features for using above formula.
Yet in practice, it turns out above formula / algorithm works fine wheter these features are independent to each other or not.



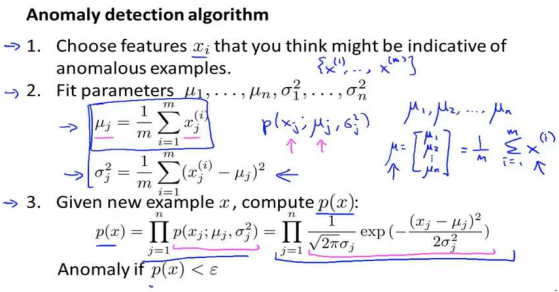
2. Anomaly detection algorithm

>1. Choose features x_i that might be indicative of anomalous examples;
choosing features that describe general properties of examples collected.

>2. Collecting training set: unlabeled data: x(1), ...x(m).

>3.Estimate parameters u_1, ..u_n, 6_1^2, ...6_n^2 for each feature gaussian density parameter based on training set:
u_i = average of x_i feature of all examples;
6_i ^2= average variance of x_i feature from u_i;

>3. Given new example x (test/dev) -->compute p(x (test)),
Anomaly if p(x_test) < ε
(note: still mean and scalling features first?no necessary?)



3. Anomaly detection example

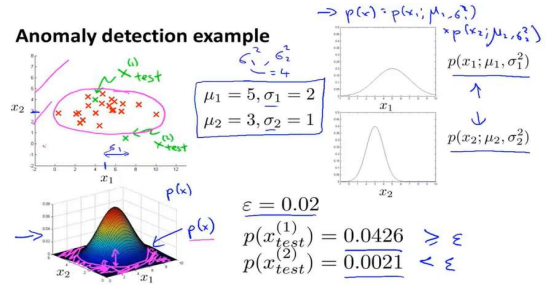
Right pic. Example:
training data have two features, x_1, x_2, --->
>1. get each feature's gaussin density, and p(x_1), p(x_2), and example desentify : p(x(i)) = p(x_1)_i * p(x_2)_i

>2. Ploting of p(x): give x_1, x_2, plotted surface hight is the p(x)

>3. Getting decison boundary : p(x) = ε

for new example: x_1 test, x_2_test:
method 1: compute probabilty;
P(x_1_test) >ε;
P(x_2_test) <ε;-->anomal

Method 2: just check if x_1 test, x_2 test in decison bundary or not.



16_118. Developing and evaluationg an anomaly detection system_Anomaly detection

Purpose this item:
1. Developing a specific application of anomaly detection to a problem
2. Evaluate an anomaly detection algorithm.

1. The importance of real-number evaluation:

>1. Real-number evaluation idea:
When developing a learning algorithm (choosing features, etc.), making decisons is easier if we have a way of evaluating on learning algorithm that just gives you back a number.

eg. : have an idea for adding one extra feature---> run the algorithm with the feature, and without the feature ---> and get back a number that tells did it improve or worsen performance to add this feature.
Then it gives a much better, simpler way, with which to decide whether or not to include that feature.

The importance of real-number evaluation

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

→ Assume we have some labeled data, of anomalous and non-anomalous examples. ($y = 0$ if normal, $y = 1$ if anomalous).

→ Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ (assume normal examples/not anomalous)

→ Cross validation set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

→ Test set: $(x_{test}^{(1)}, y_{test}^{(1)}), \ldots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

y=1

> 2. Evaluation for anomal detection algorithm:
we'll be treating anomaly detection as an unsupervised learning problem, using unlabeled data.

Standard way of evaluationg anomaly detection system:
>>1. Assume have some labeled data, of anomalous and non-anomalous examples (y=0 if normal, y=1 if anomalous)

Process of developing and evaluating an anomaly detection algorithm:

>>> 1. Training set : x(1), ...x(m), still unlabeled (assume normal or not-anomalous)
Note: even if a few anomalous examples slip into unlabeled training set,

>>> 2. Define dev and test set: to evaluate anomaly detection algorithm:
for both dev and test set, assume we can include a few examples that known as anomalous,(dev, test set are labeled)
Dev set: (**x(1)_cv, y(1)_cv**) , ... (x(m)_cv, y(m)_cv);
Test set: (x(1)_test, y(1)_test) , ... (x(m)_test, y(m)_test);

2. Aircraft engines motivating examples

data set:
10,000: good (normal) engines -does not matter if few anomalous examples slipped in.
20: flawed engines (anomaous)

>2.1 Typical Data splittying:
normal data: split as 60%, 20%, 20% to training, dev, test;
anomaous data: split: 0 %, 50%, 50% to training , dev, test set.

Not recommend alternative splitting method: use same norma/ anomaous data in dev and test set.

>>1. training set: 6000 , normal
note: still call unlabeled data set even we already know they are all normal (y=0).
-->Fit p(x)=p(x_1) x ... p(x_n) parameter (u, variance fitted training set)

Aircraft engines motivating example

→ 10000 good (normal) engines
→ 20 flawed engines (anomalous) 2-50 y=1
→ Training set: 6000 good engines (y=0) $p(x) = p(x_1; \mu_1, \sigma_1^2) \ldots p(x_n; \mu_n, \sigma_n^2)$
CV: 2000 good engines ($y = 0$), 10 anomalous ($y = 1$)
Test: 2000 good engines ($y = 0$), 10 anomalous ($y = 1$)

>>2. Dev set: 2,000 normal (y=0) + 10 anomaous (y=1)
>>3. Test set: 2,000 normal (y=0) + 10 anomaous (y=1)

> 2.2 Principle: develop and estimate anomal detect system on dev set:
on dev/ test set, think of the anomaly detention algorithm as making prediction for these y labeled in dev and test set..:
very similar to supvervised learning algorithm: have label test set and algorithm is making predictions on these labels and evaluate it by seeing how often it gets these labels right.

y=1 if p(x) < ε (anomaly)
y=0 if p(x) >= ε (normal)

note: ε is hyper perameter, could use dev set to choose ε .

> dev/ test set is very skewed, much smaller anomalous data (y=1) than normal examples.
this is much closer to the source of evaluation metrics we can use in supervised learning.
---------> Classification accuracy would not be a good evaluation metrics. as talked about ealier.

Algorithm evaluation

→ Fit model $p(x)$ on training set $\{x^{(1)}, \ldots, x^{(m)}\}$ $(x_{test}^{(i)}, y_{test}^{(i)})$
→ On a cross validation/test example x, predict

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

y=0

Possible evaluation metrics:
→ - True positive, false positive, false negative, true negative
→ - Precision/Recall
→ - F_1-score ← CV

Can also use cross validation set to choose parameter ε ←

> 2.3: Possible evaluation metrics:

>>1. computing : Ture positive, false positve, false negative, true negative;
>>2.computing: Preison / Recall;
>> 3.computing: F1 score

Above would be ways to evaluate an anomaly detection algorithm on your cross validation set or on your test set.

3. Choosing decision parameter: ε
use dev set to choose ε.

Try different ε value on dev set, and **pick the one that maximize f1 score.**

summary:
>Training set_unlabels: train algorithms;(all normal)
>Dev set_labeled: make all decisions by evaluting algoirthms on dev set:
E.g: what features to include, or trying to, tune the parameter ε.

>Test set_labeled: we can then take the final model and do the final evaluation of the algorithm on the test sets.

Note:
Here use a bit of labeled data in order to evaluate the anomaly detection algorithm and this takes us a little bit closer to a supervised learning setting.

16_119. Anomaly detection vs supevised learning__Anomaly detection

Item purpose:
Guidelines: when should probably use an anomaly detection algorithm, and whether it might be more fruitful instead of using a supervisor in the algorithm.

Note: labeled examples were used in evaluating anomaly detection system, why not just use supervised learning algorithm?

1. Conditons to use Anomaly detection and supervised learning

1. 1. Anomaly detection usage coditions:

> 1. Very skewed data set:
>>> **Very small positive examples** (y=1, anomaous examples), 0-20 /50 pc;
this small positiv examples, usually save just for dev and test set.

>>> Large negative examples (y=0):
fit the model p(x) on negative exmaples.

Anomaly detection applications idea:
Have very few positive examples and lots of negative examples;
and fitting all those Gaussian parameters in model only to negative examples.
---> if have a lot negative data, we can still fit p(x) pretty well.

Anomaly detection	vs.	Supervised learning
Very small number of positive examples ($y = 1$). (0-20 is common).		Large number of positive and negative examples.
Large number of negative ($y = 0$) examples. $p(x)$		
Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.		Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set.
		Spam

>>> Many different ' types ' of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.

it can be difficult for an algorithm to learn from small set of positive examples what the anomalies look like. And in particular, future anomalies may look nothing like the ones you've seen so far. So maybe in your set of positive examples, you've seen 5 or 10 or 20 different ways that an aircraft engine could go wrong. But maybe tomorrow, you need to detect a totally new set, a totally new type of anomaly. And if that's the case, it might be more promising to just model the negative examples with alnomaly detection algorithm, as hard to model the positive examples.

1. 2. Supervised learning conditions:

> 1. Large number of positive and negative examples:

>2. Enough positive examples for an algorithm to get a sense of what the positive examples are like, future positive examples are likely to be similar to ones in the training set;
--> More reasonable to have a supervisor in the algorithm that looks at all of the positive examples, looks at all of the negative examples, and uses that to try to distinguish between positives and negatives.

1.3 Summarize:
for using anomaly detection or supervised learning:
key difference really is: a small number of positive examples
In anomaly detection,
it is not possible for a learning algorithm to learn that much from small number of positive examples.
so we instead take a large set of negative examples and have it learn p(x) model from just the negative examples.
and Reserved the small number of positive examples for evaluating algorithms to use in the either dev set or test set.

Note: for previous e-mail spam probelm: have large set of examples of spam, -->take it as supervised problem even have many types/feature of spam.

2. Applications of anomaly detection and supervised learning:

2.1: Anomaly detection:

>1. Fraud detection:
note:
if you actually have a lot of examples of y=1, then sometimes fraud detection could actually shift over to the supervised learning column.
But, if you haven't seen that many examples of users doing strange things on your website, then more frequently fraud detection is actually treated as an anomaly detection algorithm rather than a supervised learning algorithm.

>2. Manufacturing
if manufacture in very large volumes and see a lot of bad examples, maybe manufacturing can shift to the supervised learning column as well.
But if you haven't seen that many bad examples then do the anomaly detection monitoring machine.

>3 Moinitoring machines in a data center
Similar sorts of arguments to apply.

Anomaly detection	vs.	Supervised learning
• Fraud detection $y=1$		• Email spam classification
• Manufacturing (e.g. aircraft engines)		• Weather prediction (sunny/rainy/etc).
• Monitoring machines in a data center		• Cancer classification
⋮		⋮

2.2: Supervised learning:
>1. Email spam classification;
>2. Weather prediction;
>3. Cancer classification.

If have **equal,** many numbers of positive and negative examples, then we would tend to treat all of these as supervisor problems.

Summarize:
Above gives one of a learning problem properties that would cause to treat it as an anomaly detection problem versus a supervisory problem.
For many other problems that are faced by various technology companies and so on, we actually are in the settings where we have very few or sometimes zero positive training examples. There's just so many different types of anomalies that we've never seen them before. And for those sorts of problems, very often the algorithm that is used is an anomaly detection algorithm.

16_120 Choosing what features to use__Anomaly detection

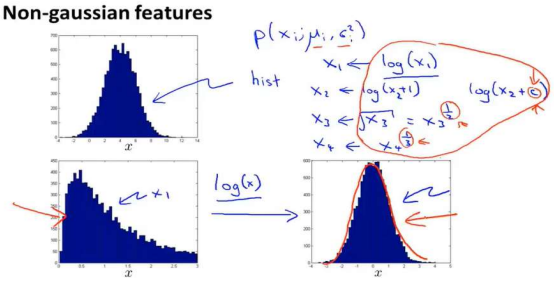
Features used in anomaly detection have huge impact on algorithm performance
Item prupose: give Guidline for designing or selecting features.

1. Non-gaussian features

In anomaly detection: model each features using Gaussian distribution.

>1. Plotting data /hisogram （直方图） of the data before feeding to anoamly detectin, make sure it actually looks vaguely Gaussian.
in cases the data looks non-Gaussian, the algorithms will often work just fine.
But if it doesn't look at all like a bell shaped curve-Gaussian density curve,
---->
play with different transformations of the data in order to make it look more Gaussian.
And again the algorithm will usually work okay, even if you don't. But if use these transformations to make your data more gaussian, it might work a bit better.

Non-gaussian features



Note : the way to plot a histogram is to use the HIST, or the HIST command in Octave:
horizental axile: feature value; vertical axile: example numbes corresponds to a specified feature value.

>2. Data transformation method: how does it works??
log (x): e.g replace feature x_1 by log (x_1): x_1' = log (x_1)
log (x+c) : c: constant used to make this new generated feature more like Gaussian
x^c: c: constant used to make this new generated feature more like Gaussian

Summarize:
> 1. Plot a histogram with the data/one or each feature;
>2.if the plotting looks pretty non-Gaussian, it's worth playing around a little bit with different transformations, to see if can make data look a little bit more Gaussian, before feed it to learning algorithm.
although even if you don't, it might work okay. But I usually do take this step**. how works ok??**

2. Error analysis for anomaly detection:

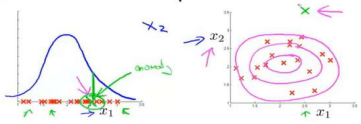
Come up with features for an anomaly detection algorithm, by error analysis procedure, which is really similar to the error analysis procedure for supervised learning.

2.1 Supervised learning error analysis procedure:
>1. Train a complete algorithm,
>2. Run algorithm on a cross validation set;
>3. look at the examples it gets wrong on dev se---> see if we can come up with extra features to help the algorithm do better on the examples that it got wrong in the dev set.

→ Error analysis for anomaly detection

Want $p(x)$ large for normal examples x.
$p(x)$ small for anomalous examples x.

Most common problem:
$p(x)$ is comparable (say, both large) for normal and anomalous examples



2.2 Anomaly detection error analysis procedure:

P(x) Expectation: is large for normal example and small for anomalous example

Common probelm (found in dev set): in dev set, some p(x) also large for anomalous example.
/ p(x) is comparable (say, both large) for normal and anomalous examples.
algorithm fails to flag this as an anomalous example.

-->look at training examples and look at what went wrong with that particular dev example, and see if looking at that example can inspire to come up with a new feature x2, that helps to distinguish between this bad example, compared to the rest of normal/training examples. then train anomaly detection algorithm on new training examples with new features.

re-plot training example and new example with new features,
-->new example -anomaous example should be outside decision boundary;
or
--->train gaussian algorithm with new training data (with new features), compute p(x_test) should < ε.

3.Creating new features:

concerns while choosing features:
choose features that will take on either very, very large values, or very, very small values, for examples that might turn out to be anomalies.

E.g: monitoring computers in a data center:
have alreay choosen right pic, four features. but now suspect one failure case: computers has a job that gets stuck in some infinite loop, cause high CPU load, and low network traffic-

-->create new feature x_5= CPU load / traffic. will be unusually large value when have that suspected error.

By creating features like these, you can start to capture anomalies that correspond to unusual combinations of values of the features.

→ Monitoring computers in a data center

→ Choose features that might take on unusually large or small values in the event of an anomaly.
- → x_1 = memory use of computer
- → x_2 = number of disk accesses/sec
- → x_3 = CPU load
- → x_4 = network traffic

$x_5 = \frac{\text{CPU load}}{\text{network traffic}}$ $x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$

Summarize:
This term: talked about
> how to and take a feature, and maybe transform it a little bit, so that it becomes a bit more Gaussian, before feeding into an anomaly detection algorithm.
>And also the error analysis in this process of creating features to try to capture different types of anomalies.
And with these sorts of guidelines hopefully that will help you to choose good features, to give to your anomaly detection algorithm, to help it capture all sorts of anomalies.

16_121 Multivariate Gaussain distribution_Anomaly detection

Multivariate Gaussain distribution: can sometimes catch some anomalies that the earlier algorithm didn't

1. Gaussian distribution problem:

for features x_1, x_2, ...x_n:, Gaussian density decision boundary is :
(x_1- u_1)^2 / б_1 ^2 + ...+ (x_n- u_n)^2 / б_n ^2 =c
c: decided by б_i and ε

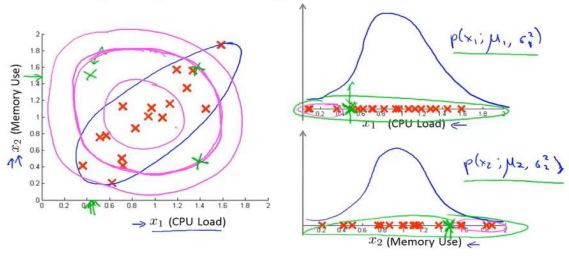
>Decison boundary is symmetric about feature axile (features are vertical to each other / independent to each other)
when features are highly relevant-e.g linear relateed- this symmetric decision boundary could not fit training example well.

2. Motivating example: monitoring machines in a data center:

Training set: feature x_1, x2 is linearly related

-->Gaussian density decison boundary could not fitt training set well:
Fail to distinguish the anomaous example-green point: as now p(x_1) and p(x_2) is still ok for green point.

Motivating example: Monitoring machines in a data center



3. Multivariate Gaussian (Normal) distribution

Principle:
> X in R^n
> Do not model p(x_1), p(x_2), ..p(x_n) separately
> model p(x), all in one go: model p(x) all at the same time.

Parameters: u, Σ (decided by training set)
>1. u: vector in R^n,
>2. Σ: (n,n) matrix; covariance matrix Σ = 1/m * sum (x (i) * x(i)^T) = 1/m *X * XT over all m training example.
(note: X = [x1, x2, ..xm])
Similar to the covariance matrix that we saw when we were working with the PCA, with the principal components analysis algorithm.

Model: p(x)
|Σ| ^1/2: determinant of Σ （行列式）: command used in Octave: det (Sigma)

Multivariate Gaussian (Normal) distribution

$x \in \mathbb{R}^n$. Don't model $p(x_1), p(x_2), \ldots$, etc. separately.
Model $p(x)$ all in one go.
Parameters: $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix)

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

$|\Sigma|$ = determinant of Σ | det (Sigma)

4. Mulitvariate Gaussian examples:

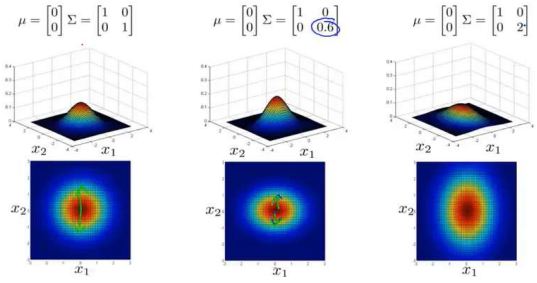
eg.: have only features x_1, x_2
4.1 Independence features:
e.g 1: u=0;0]T, Σ= [1, 0; 0,1]
>p(x): highest/peak value when x_1=x_2=0
>contour plot(feature boundary for specified p(x)) : symmetric about axile.
p(x) goes down from plot center to outside point

e.g 2: u=0;0]T, Σ= [0.6, 0; 0,0.6]: shrink Σ
>p(x): peak value increase, and curve width smaller (area below p(x) surface is 1).
>contour plot(feature boundary for specified p(x)) : symmetric about axile, and shrink

e.g 3: u=0;0]T, Σ= [2, 0; 0,2]: increase Σ
>p(x): peak value decrease, and curve width bigger (area below p(x) surface is 1).
>contour plot(feature boundary for specified p(x)) : symmetric about axile, and flatter

e.g 4: u=0;0]T, Σ= [2, 0; 0,1]: increase only one feature variance
>p(x): peak value decrease, and curve width bigger (area below p(x) surface is 1) in x_1 direction
>contour plot(feature boundary for specified p(x)) : symmetric about axile, and flatter in x_1 direction

Multivariate Gaussian (Normal) examples



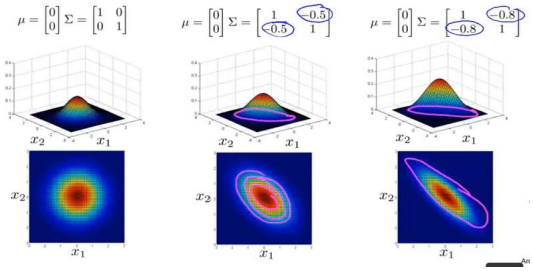
4.2 Model correlations between data/feature
feature x_1, x_2 are highly correlated with each other.

e.g 1: u=0;0]T, **Σ= [1, 0.5; 0.5,1]: features highly correlated**
>p(x): highest/peak value when x_1=x_2=0
>contour plot(feature boundary for specified p(x)) : **asymmetric** about axile, but instead symmetric algong y=x
p(x) goes down from plot center to outside point

e.g 2: u=0;0]T, **Σ= [1, 0.8; 0.8,1]: increase variance**
>p(x): more peak along the y=x line (shrink vertical to y=x line): flatter along y=x direction
>contour plot(feature boundary for specified p(x)) : asymmetric about axile, but instead symmetric algong y=x
x_1, x_2 tend to be large together.

e.g 3: u=0;0]T, Σ= [1,-05; -0.5,1]: negative relavent
>p(x): highest/peak value when x_1=x_2=0
>contour plot(feature boundary for specified p(x)) : asymmetric about axile. symmetric along y=-x;

Multivariate Gaussian (Normal) examples



Note: change u only change distribution location, do not change p(x) value.

16_122 Anomaly detection using the multivariate Gaussain distribution_Anomaly detection

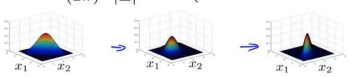
1. Mulitvariate Gaussian (Normal) distribution:

Training set : {x(1), ...x(m)};

Parameter fitting:
u= average of x(i) (n,1) matrix
Σ: same as in PCA (n, n) matrix

Multivariate Gaussian (Normal) distribution
Parameters μ, Σ $\quad \mu \in \mathbb{R}^n \quad \Sigma \in \mathbb{R}^{n \times n}$

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



Parameter fitting:
Given training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $\quad x \in \mathbb{R}^n$

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

2. Anomaly detecion with the multivariate Gaussian

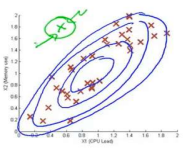
>1. fitting parmaters u, Σ on training set;
>2. Model p(x);
>Given new example x_dev: check if p(x) < ε

e.g: for right pic , anomalous example-green point, multivariate Gaussian will flag it an anomaly sucesfuly.

Anomaly detection with the multivariate Gaussian

1. Fit model $p(x)$ by setting

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



2. Given a new example x, compute

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

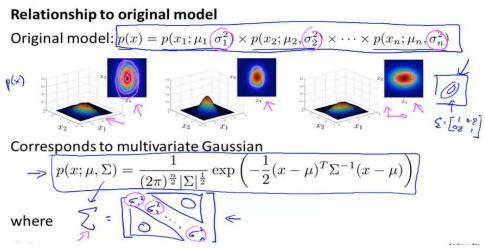
Flag an anomaly if $p(x) < \varepsilon$

3. Relationship to original model

>1. Original model: p(x) is the product of p(x_1, u_1, 6_1^2) * ..p(x_n, u_n, 6_n^2);

>2. Original model, corresponds to multivariate Gaussian where the contours of the Gaussian are always axis aligned （等高线轴对称）:means:
Σ must have 0 on the off diagonal elements, and put 6_1^2, ...6_n^2 in diagonal elements
can not model correlations between diffrent features.

>3. Original model: contours of the Gaussian is axis-feature algihed.
while Mulitvariate Gaussian axis of Gaussian contour has an angle.



4. Conditions to use Origianl Gaussian and mulitivirate Gaussian

The original model is probably used somewhat more often, and whereas the multivariate Gaussian distribution is used somewhat less but it has the advantage of being able to capture correlations between features.

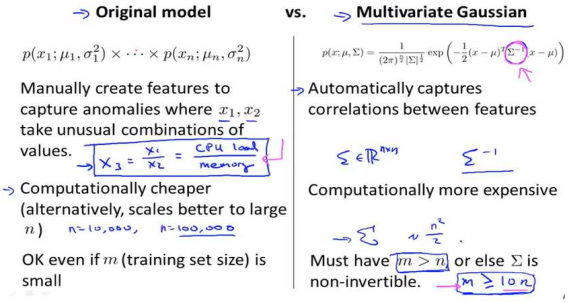
4.1. Original model

> 1. Extra new feature needed to caputure feature combination value.

while using original model, need to create extra features if there's unusual combinations of some features, even though these feature themselves looks like it's taking a perfectly normal value.
-->in this case original model works fine.
(note: but in this case , new feature created by combining other features, is releated with other features, distribution not independent to other features..)

>2. Computationally cheaper (alternatively , **scales better to large feature number n**)
Even if n were ten thousand, or even if n were equal to a hundred thousand, the original model will usually work just fine.

>3. work out ok even if have a relatively small training set
This is the small unlabeled examples that we use to model p of x



4.2 Multivariate Gaussian

>1. automatically capture correlations between different features.

>2. Computationally more expensive:
need to compute the inverse of the matrix sigma where sigma is an n by n matrix.
while computing sigma if sigma is a hundred thousand by a hundred thousand matrix that is going to be very computationally expensive.
--->multivariate Gaussian model scales less well to large values of n.

>3. Must have m >n , or else Σ is non-invertible.-mathmatic property
if m<n , Σ is not invertible, could not use mulitvirate Gaussian model.
>>typical rule of thumb for using multivariate Gaussian model:
only if m is much greater than n: e.g m>=10*n:
to make sure have enough data to feed to Σ which have n^2 number parameters. (enough example to get features x1, ..xn relationship

Summarize:
1 . in practice the original model is used more often.
And if suspect need to capture correlations between features, just manually design extra features to capture specific unusual combinations of values.

2. But when have a very large training set or m is very large and n is not too large, then the multivariate Gaussian model is well worth considering and may work better as well.

And can save from having to spend your time to manually create extra features in case the anomalies turn out to be captured by unusual combinations of values of the features.

Note:
if Σ is signluar (chances in pratice is very low), check:
1. traing set size,f ensure number m >n or not?
2. if there is reducdant feature: e.g: x_2=x_1; x_3=x_2+x_1;..(note: this relationship is multivariant model gonna to capture...)

W17 - Recommender Systems

17_125. Problem formulation_Recommender Systems

Recommender system:

Motivations for talking about recommender systems:
>1. it is an important application of machine learning, for company
An improvement in performance of a recommender system can have a substantial and immediate impact on the bottom line of many of these companies.

Note:
Even recommender systeem ist less paied attenion within academic , or at least it's sort of a smaller fraction of what goes on within Academia. But for many technology companies, the ability to build these systems seems to be a high priority.

>2. Big ideas in Machine learning:
for some problems, there are algorithms (recommender system is one of them) that can try to automatically learn a good set of features.

1. Example: predicting movie rating:

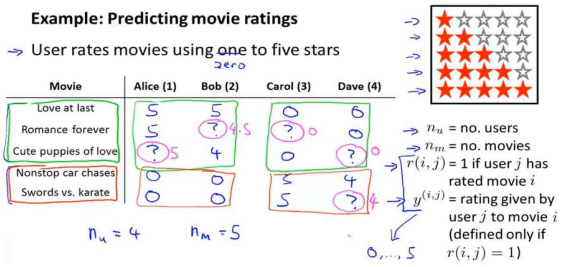
1.1: Definations:

n_u= number of users: 4;
n_m =nimber of moviews: 5;
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).

each user rate/do not rate all the movies.

--->Recommender system problem:
Come up an algorithm, that can **automatically** predict the user j's ratting on the movies he/she has not rated , based on the rating he /she and other users' rating on these movies ((Given y(i, j), r(i,j))

e.g: in right pic, examge: based on four users rating on five movies, try to predict their rating on the movies they have not rated. then could recommand movies to the users.



16_126. Content-based recommendations_Recommender Systems

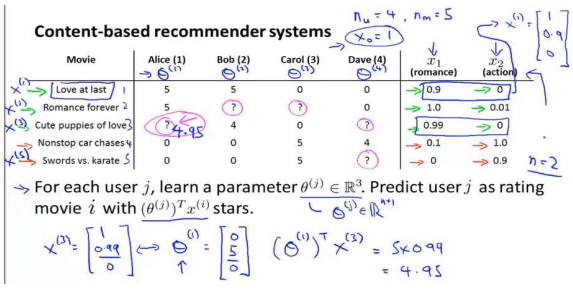
1. Content-based recommender systems

Data:
>1. Have all movies' feature vector: x (i): [1; x_1; x_2]T already known;
e.g: movie relevent fraction to feature 'roman', 'action'.
Note: x_0 = 1

>2. Have user's rate to leas some movies in step one (whose feature already know)

>For each user, learn a parameter vector θ_j in R^3, based on data in step 1 and step2.
θ_j: user's favorite to movie's each feature.

> 4. Predict user j as rating movie i with θ_jT* x(i) stars
actually is appling a copy of linear regression for each user.

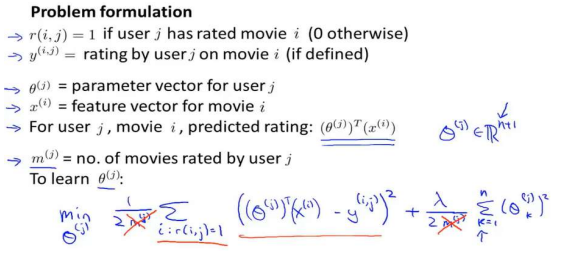


2. Problem formulation

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);
x(i) = movie feature vector;
θ (j) = user parameter vector

for user j, movie i, predicted rating: θ (j)T* x(i).
m(j) = number of movies rated by user j.

Learn θ (j) : linear regression problem:
(note: not use softmax function, as rating is continues values, not discrete.)
min (θ (j)T* x(i) -y(i,j))^2 sum over all movies m(j) + regulization term (θ_0 no considered here; sum over moive feature number size n)
(note: removed dividing 1/m(i,j) here, no influence on object optimization result)



3. Optimization objective:

>1. Learn all users' parameter,:

objective: just sum over each user's optimization objective. And minimize it for all users' parameter vector.

Min J (θ (1), ..θ (n_u)

finding each user parameters that minimizing devation of user real rating and predicted rating with computed user parameter.

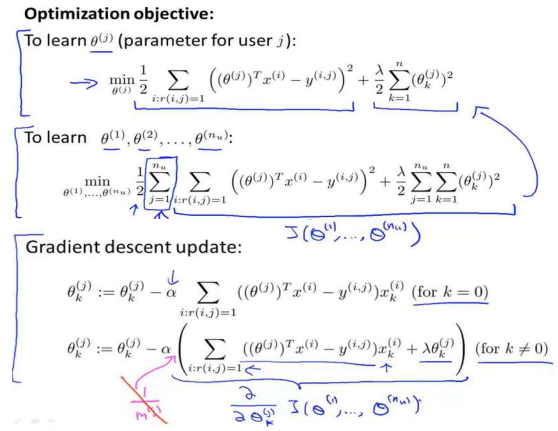
(note: each user's parameter vector θ actually optimized only by it's own loss, seperately-each user's parameter updaing and rating do not influence other's parmaeter optimizaiton process and result)

>2. Gradient descent update:

sperate formula for k=0, and k/=0), as θ(i)_0 not considered in regulization
θ (j)_0: do not condier regulization in optimizaiton objective;
θ (j)_k :consider regulization in objective

only difference with linear regression is removed '1/m' in optimization objective, which do not influece optimization result.

above formulas could be used for the derivative while can also plug them into a more advanced optimization algorithm, like conjugate gradient or LBFGS or what have you. And use that to try to minimize the cost function J as well.



Summary:

Above algorithm is called a content based recommendations, or a content based approach:
because we assume havinge available features for different movies , which caputer movies content (how romantic, how much action is in the movie). And using movie content features to make our prediction.

16_127. Collaborative filtering__Recommender Systems

1. Problem motivation:
Hard to get movie feature:
it's very difficult and time consuming and expensive to actually try to get someone to, watch each movie and tell you how romantic each movie and how action packed is each movie, and often you'll want even more features than just these two. So where do you get these features from?

Collaborative filtering:
has a very interesting property -feature learning: learn for itself what features to use.

Example: Predict movie featurs based on user rating and user parameter

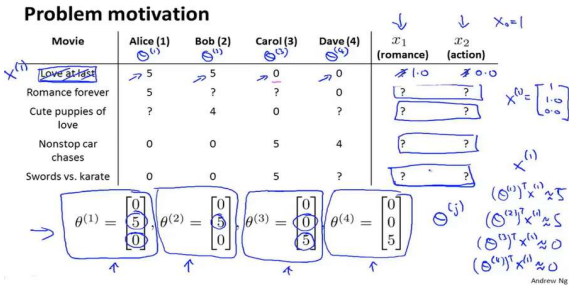
Data availabel:
>1. User rating on at least some movies. Movie feature no idea what is .
>2. User paramter θ(j): favourate for different stype of moives.
E.g: survey done by customer for their favorite on moives-->parameter vector.

Note: θ(j)_0=0, while movie feature x(i)_0 =1.

e.g:
x(1) T * [θ(1), θ(2), θ(3), θ(4)] =[5, 5, 0,0] --->x(1)_1=1, x(1)_2=0
x(1) T =[1, x_1, x_2]

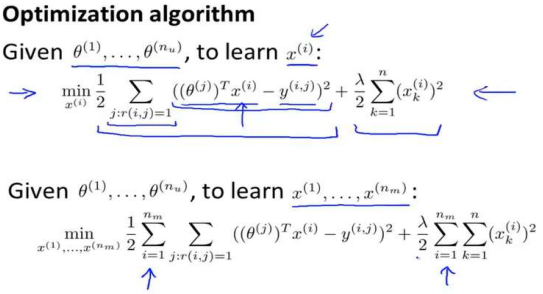
θ(1)_1 = θ(2)_1=5, means users favoriate on love movie
-->move x(1)_1 feature corresponds to 'romance' feaure,
x(1)_1=1-->movie x(1) is romance movie.

same way to figure out other movies' feature.



2. Optimization algorithm

>2.1 for one movie:
Given user parameter θ(1), θ(2), θ(3)... θ(n_u), and their rating y(i, j) on movie x (i),
to learn x(i):
min J: for movie x(i), finding it's feature that minizing **variation of all users' real rating on this movie** and predicted rating with feature computed.
sum $(\theta(j)^T x(i) - y(i, j))^2$ over all users + **regulization about x(i)_k (k=1, ...n: movie feature number)**



note:
while find one user paramter, optimization algorithm is:
finding each user parameters that minimizing **variation of this user's real rating on all movies** and predicted rating with computed uer parameter.

>2.2 for all moives;
objective: sum one moive's objective over all moives

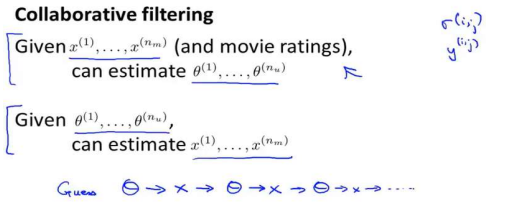
3. Collaborative filtering:

Algorithm 1:
Given movies' feature vector: x(1), ...x(m), (and users' movie rating)--->estimate user paramter θ(1), θ (2), θ(3)... θ(n_u)

Algorithm 2:
Given user paramter θ(1), θ(2), θ(3)... θ(n_u), (and users' movie rating)--->estimate movies' feature vector: x(1), ...x(m)

Collaborate filtering:

get initia θ--->get X through algorithm 2 ------>update θ with algorithm 1------>update x with algrothm 2.....
will converge to a reasonalbe movie feature x and user parameter θ.



Note:
For the recommender system problem, this is possible only because each user rates multiple movies and hopefully each movie is rated by multiple users. And so you can do this back and forth process to estimate theta and x.

Summarize:
Collaborative filtering refers to the observation that when run this algorithm with a large set of users, what all of these users are effectively doing are sort of collaboratively--or collaborating to get better movie ratings for everyone.
Because with every user rating some subset with the movies, every user is helping the algorithm a little bit to learn better movie features, and then by helping-- by rating a few movies myself, I will be helping the system learn better movie features and then these features can be used by the system to make better movie predictions for everyone else (note: better movie feature helps to learn users parameters).
And so there is a sense of collaboration where every user is helping the system learn better features for the common good.

16_128. Collaborative filtering algorithm__Recommender Systems

1. Collaborative filtering optimizaiton objective

Collaborate filtering principle: update parameter θ, X sequently.

get initia θ--->get X through algorithm 2 ------>update θ with algorithm 1------>update x with algrothm 2.....

>1. Motivation:
More efficient algorithm: to update parameter θ, X simultaneously:
Take Collaborate filtering algorithm object as below:

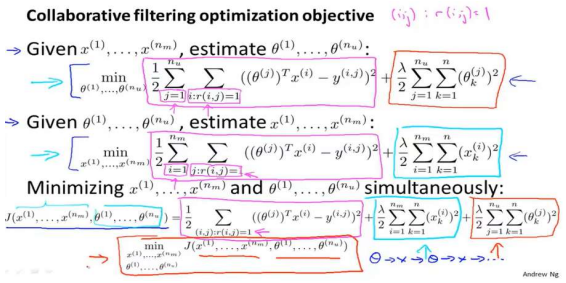
combine last item: algorithm 1 & algorithm 2 objective together.
>1. First item:
= algorithm 1 = algorithm 2.
algorithm1: sum over all users j and then sum over all movies rated by that user.
algorithm 2: sum over all movies, and then sum over all the users j that have rated that movie;
collaborative filtering: sum over all pairs(i, j) for which r (i, j)=1. It's just something over all the user movie pairs for which you have a rating.

>2. Second item: regulization
regulization of parameteer θ only (in algorithm 1) + regulization of x only (in algorithm 2).

>2.Objective:
min J(θ, x) + θ regulization + x _regulization
θ regulization =0 , when computing partial derivation of x;
x regulization =0 , when computing partial derivation of θ;

-->partial derivation of objective over θ = aglorithm 1 objective derivative over θ;
-->partial derivation of objective over x = aglorithm 2 objective derivative over x;

Note: in algorithm1, 2, and new version object above, do not consider x(i)_0, θ(j)_0.
do away with this convention is: because algorithm now has the flexibility to just learn it by itself. if the algorithm really wants a feature that is always equal to 1, it can choose to learn one for itself. So there's no need to hard code the feature to 1.



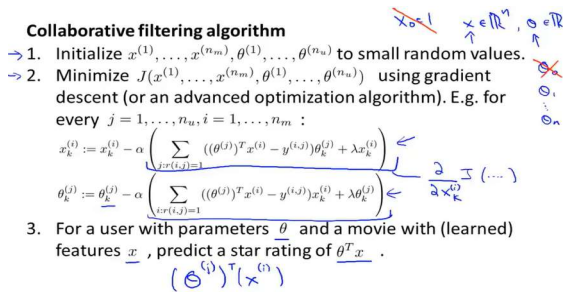
Note:
only difference between collaboriative filtering algorithm objective and the older algorithm is :
Instead of going back and forth, **sequentially minimizing with respect to θ then minimizing with respect to x, in** this new version, just **minimize both sets of parameters simultaneously.**

2. Collaborative filtering algorithm

>1. Initialize x and theta θ to small random values.
A little bit like neural network training, where there we were also initializing all the parameters of a neural network to small random values.
>2. Minimize the cost function using gradient descent or one of the advance optimization algorithms.

Note:
no longer have this x(i)_0 = 1, and no longer the special case θ_0 (not inclued in objective), which was regularized differently,
---> gradient descent no break out a special case for k=0.

>3. For user with (learned) parameter θ and movie with (learned) feature x, predict a star rating of θT* x.
e.g: if user j has not yet rated movie i, then predict that user j rate on this i movie by $\theta(j)^T * x(i)$



Summary:
Collaborative filtering algorithm will simultaneously learn good features for hopefully all the movies as well as learn parameters for all the users, and hopefully give pretty good predictions for how different users will rate different movies that they have not yet rated.

16_129. Vectorization: low rank matrix factorization__Recommender Systems

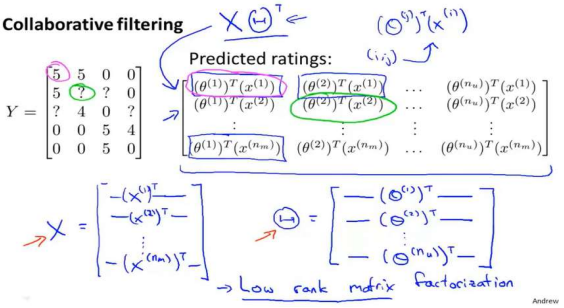
Item purpose:

1. vectorization implementation of collaborative filtering algorithm;
2. Recommend product to users that has recently been looking at one product.

1. Collabrative filtering:
-->work out an alternative way of writing out the predictions of the collaborative filtering algorithm.
Take all the ratings by all the users and group them into a matrix.
X^T=[x(1), x(2) ... X(n_m)]
Θ^T =[θ(1), θ(2),...θ(n_u)]

Predicted raing: $X * \Theta^T$

Collaborative filtering algorithm --->called Low rank matrix factorization (低秩矩阵分解): $X * \Theta^T = Y$



2. Finding related movies

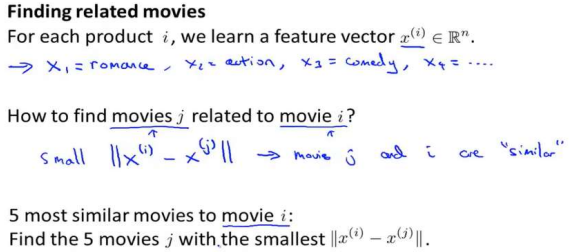
Having run the collaborative filtering algorithm , could use the learned features to find related movies.

> for each product x , have learned a feature vector x(i) in R^n.

when learn a certain features, do not know in advance what the different features are going to be, but if run the algorithm and perfectly the features will tend to capture what are the important aspects of these products that cause some users to like certain movies and cause some users to like different sets of movies.

for these learned features/important aspect of products, it's actually often pretty difficult to come up with a human understandable interpretation.

> Find product j related to product i:



~~3. Find product j related to product i.~~
so as to recommed related product to users to buy.

similarity of two products that have learned their features x(i), x (j):
use feature vector distance ||x(i) - x(j) ||

Summary:

Use a vectorized implementation to compute all the predicted ratings of all the users and all the movies, and also do things like use learned features to find what might be products that are related to each other.

16_130. Implementational detail: Mean normalization Vectorization: low rank matrix factorization__Recommender Systems

1. Users who have not rated any movies

e.g: in right pic, user_5 have not rated any movie , try to find user parameter θ(5):

Optimization objective: first term now do not play a role and no rating y(i,j), so only second term regulization of θ(5) parameter.
-->θ(5) =0 vector

-->for any movie x(i), user_5 rating is $\theta(5)^T * x$ =0

Problem:
This is not not reasonable or useful :
Some people do like some movies, yet we could not recommend any movies to this user as rating of this user for all moviews is 0.

2. Mean normalization:

>1. get avearage rating for all movies u vector (note: for each movie; without new user involvoed/counted).

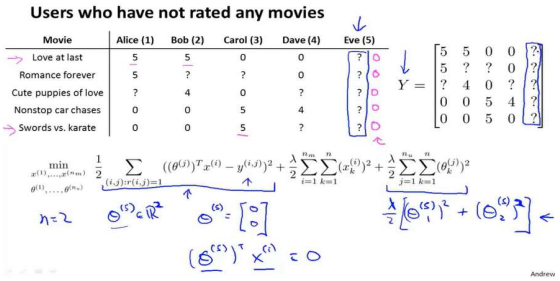
>2. mean normailzation data set:
updated rating matrix Y: = Y-U , including new user rating also in Y(which is unknow);

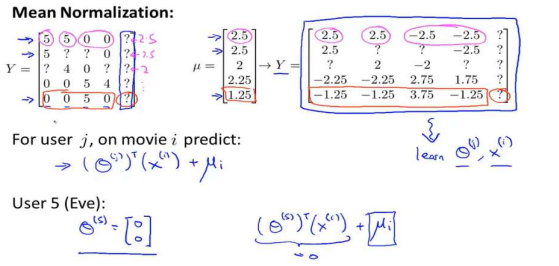
>3. Pretend rating Y in step 2 is the actual ratings from users, to use in collaborative filtering algorithm, to get paramter θ(1), ..θ(5), and x(1), ...x(5).

>4. For user j, on movie i predict: $\theta(j)^T * x(i) + u$

-->for new user_5, θ(j)=0, yet rating on movies will be mean rating: U
This actually makes sense, because it says that if new user hasn't rated any movies and we just don't know anything about this new user , what we're going to do is just predict for each of the movies, what are the average rating that those movies got.

(coding: for Y matrix: movies has no rating: ?:-how to make in coding do not involve them in computation?)





3. Movies have no ratings:

In case some movies with no ratings, can also play with versions of the algorithm. yet only normalize the different columns to have means zero, instead of normalizing the rows to have mean zero.

although that's maybe less important, because if you really have a movie with no rating, maybe you just shouldn't recommend that movie to anyone, anyway. And so, taking care of the case of a user who hasn't rated anything might be more important than taking care of the case of a movie that hasn't gotten a single rating.

W18 - Large scale machine learning

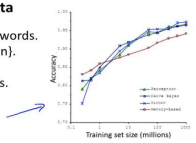
18_132. Learning with large datasets_Large scale machine learning

One of the reasons that learning algorithms work so much better now than even say, 5-years ago, is just the sheer amount of data that we have now and that we can train our algorithms on.

Machine learning and data

Classify between confusable words.
E.g., {to, two, too}, {then, than}.

For breakfast I ate ____ eggs.



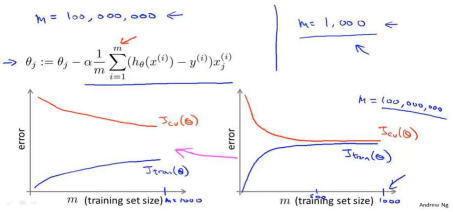
→ "It's not who has the best algorithm that wins.
It's who has the most data."

1. Machine learning and data

We've already seen that one of the best ways to get a high performance machine learning system, is if you take a **low-bias learning algorithm**, and **train that on a lot of data**.

early example-in right pick, leed to the saying in machine learning that often it's not who has the best algorithm that wins. It's who has the most data.

Learning with large datasets



2. Learning with large datasets

Learning with large data sets comes with its own unique problems, specifically, **computational problems**.

Gradient descent:
for a signle gradient descent step:
need to go through all the training examples for computing derivative term

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

if m = 100,000,000 (pretty realistic for many modern data sets), and to train a linear regression model / maybe a logistic regression model, in which case this is the gradient descent rule. then for a singal gradient descent step, the computation is expensive.

3. Smaller training set

Try: train algorithm on **smaller examples** which randomly pick from the large examples. **Might do as well as on large example.**

--> **Using ploting curve method**: error vs m (training set size): to see if adding example size will improve algorith performance (dev error mainly) effectively.

if algorithm dev and train error looks like left pic (big gap between J_train and J_dev.: a high-variance, then adding extra training examples would improve performance.
(note: overfitting: fit training set well, may learned 'fake' feature , that not applicable on dev/test set.)

if algorithm dev and train error looks like right pic. (J_train close J_dev, both big value): classical high-bias learning algorithm.
Increasing m will not do much better and then it is just fine sticking to m=1000/500
and one natural thing to do now would be to add extra features, or add extra hidden units to neural network so that end up with a situation closer to that on the left (j_train small: at least fit training set well).

Summary:
In large-scale machine learning, we like to come up with computationally reasonable ways, or computationally efficient ways, to deal with very big data sets.
Two main ideas: Stochastic gradient descent and Map Reduce, for viewing with very big data sets.

18_133. Stochastic gradient descent_Large scale machine learning

For many learning algorithms, among them linear regression, logistic regression and neural networks, the way we derive the algorithm was:
By coming up with a cost function or coming up with an **optimization objective**, and then using an algorithm like gradient descent to **minimize that cost function.**
when have a very large training set **gradient descent** becomes a computationally very expensive procedure.
(note: gradient descent step: consider all examples loss decrease direction.)

1. Linear regression with gradient descent

h(x): [θ_0;θ_1; ...θ_n]^T*[x(1), x(2), ...x(m)]
Objective: min J :1/2/m sum (h(x(i)- y(i))^2 over all examples

>1. Gradient descent:
θ := θ -a* derivative of J over θ (go through all examples: x(1), ..x(m)
repeate gradient descent till converge/ clost to converge.

>2. Gradient descent function: right pic.
start from initialized parameters then run gradient descent different iterations. Gradient descent will take the parameters to the global minimum. So take a trajectory that looks like that and heads pretty directly to the global minimum.

note: inner loop is one step, outloop is steps /iteration to go.

Linear regression with gradient descent

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

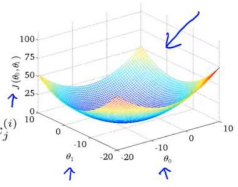
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

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

}



>3. Gradient descent probelm:
if m is large. then computing derivative term can be very expensive, as summing over all m examples in each singal gradient descent step.

>4. Batch gradient descent
This particular version of gradient descent is also called Batch gradient descent.
Batch refers to the fact that looking at all of the training examples at a time. We call it sort of a batch of all of the training examples.

Linear regression with gradient descent

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

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

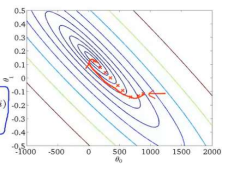
Repeat {

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

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

}

M = 300,000,000
Batch



2. Stochastic gradient descent:

Stochastic gradient descent: look at only a single training example in one iteration

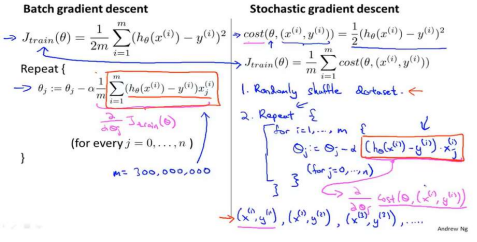
2.1 Cost function: right pic: cost (θ, x(i), y(i))
Measures how well is my hypothesis doing on a single example x(i), y(i).

2.2 Stochastic gradient descent procedure:
>1. Randomly shuffle the data set. a standard pre-processing step
In below procedure, we are visiting the training examples in some sort of randomly sorted order. Randomly shuffle the data set in practice would speed up the conversions to Stochastic gradient descent just a little bit.

>2. Inner loop:
>2.1 Update all parameters , use gradient descent with only one example (x(i), y(i)) used.
use object cost (θ, x(i), y(i)), update all parameters: [θ_0;θ_1; ...θ_n]T
(note: fit algorithm /parameters on just that one example)

>2.2 Reapet step2 with rest examples, only one example in one iteration.-scan through all examples

Stochastic gradient descent is actually scanning through the training examples.
Look at only one training example x(i), y(i), take like a basically a little gradient descent step with respect to the cost of just this training example. and then go to next training example.



>3. Outloop:
Reapet loop: step 2 & step 3.
Repeat go through stochastic gradient descent (on all examples) till parameters converge.

Note:
Batch gradient descent:
inner loop one epoch: 1 step: is one step/iteration on all example;
outloop: is the repeate time for inner loop:-->is final step/ iteration /epoch number.

Stochastick gradient descent:
inner loop one epoch: m steps: is m steps/iterations: each step /iteration use one example;
outloop: epoch times: repeate inner loop times: multipley * m is the final steps.

> **Summarize:**
Stochastic gradient descent process:
Look at the first example and modify the parameters a little bit to fit just the first training example a little bit better. Having done this then going to go on to the second training example. And what it's going to do there is take another little step in parameter space, so modify the parameters just a little bit to try to fit just a second training example a little bit better, and so on until get through the entire training set.

another view of Stochastic gradient descent:
it's a lot like bactch gradient descent, but rather than wait to sum up these gradient terms over all m training examples, stochastic gradient descent taking this gradient term using just one single training example and starting to make progress in improving the parameters already towards the global minimum.

2.3: Parameters in Stochastic gradient descent:

>1. Parameters in batch gradient descent:
Batch gradient descent algorithm looks at all the training examples in every iteration time
--->tend to take a reasonably straight line trajectory to get to the global minimum-right red line.

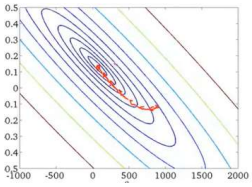
>2. Parameters in stochastic gradient descent:

> Every iteration is going to be much faster
as each iteration only look at one example instead of going through all examples.

> Parameters move generally in the direction of the global minimum, but not always.-right pink line
Because each iteration only look at one example. Each iteration try to making paramters fit only that singal training example better, iteration direction decided only by that example, which actually head in a bad direction.
And so take some more random-looking, circuitous path to watch the global minimum.

> Parametes doesn't actually converge in the same sense as Batch gradient descent does.
And ends up wandering around continuously in some region close to the global minimum but it doesn't just get to the global minimum and stay there.

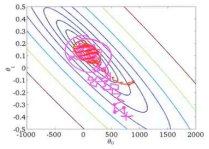
But in practice this isn't a problem because, so long as the parameters end up in some region there maybe it is pretty close to the global minimum.
So, as parameters end up pretty close to the global minimum, that will be a pretty good hypothesis and so usually running Stochastic gradient descent we get a parameter near the global minimum and that's good enough for essentially any most practical purposes.



Stochastic gradient descent

1. Randomly shuffle (reorder) training examples
2. Repeat {
   for $i := 1, \dots, m$ {
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
   (for every $j = 0, \dots, n$)
   }
   }

$m = 300,000,000$



>3. Loops to repeate stochastic gradient descent

In Stochastic gradient descent, we had this outer loop repeat inner loop multiple times.

> inner/one loop-epoch: stochastic gradient descent make parameters go 'm' steps: do every one iteration/step with one example, till go through all examples (m step).

>Outer loop: times to repeate inner loop
inner loop repeated time is depending on the size of the training set.
common/typical is and 1- 10 times. (go through entier training set 1-10 times)

With Batch gradient descent,each iteration /one little baby steps of gradient descen taking a pass through entire training set (iteration times in total), --> this is why Stochastic gradient descent can be much faster.

17_134. Mini-Batch gradient descent_Large scale machine learning

Mini-Batch gradient descent sometimes works even faster than sochastic gradient descent.

1. Mini-batch gradient descent

>1. Batch gradient descen: Use all m examples in each iteration;
>2. Stochastic gradient descent: Use a single example in each iteration;
>3. Mini-batch gradient descent: Use b examples in each iteration

Parameter: b- "mini batch size" , type choice: 2-100 (common 10)

1.1 Mini-batch idea: Somewhat in-between Batch gradient descent and Stochastic gradient descent. Rather than using one example at a time or m examples at a time/iteration we will use b examples at a time.

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 iteration

b = mini-batch size. b = 10. 2-100

Get b = 10 examples $(x^{(i)}, y^{(i)}), \dots (x^{(i+9)}, y^{(i+9)})$

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

$i := i + 10$

1.2 Mini-batch algorithm

e.g: b=10, m=1000.

>1. Inner loop:
>>> 1. one iteration: graddient descent on 10 examples
>>> 2 go continualy 100 (m/b) itertions, each iterations with next, different 10 examples
-->go thourh all examples 1 time, 100 (m/b) itertions with b example for each iteration.
each iteration use different b examples.

3. outer loop: repeat inner loop: -->go through again all the examples.- epoch.

Mini-batch faster than batch gradient descent:

Mini-batch gradient descent

Say $b = 10, m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \dots, 991$ {

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

(for every $j = 0, \dots, n$)

}

}

$m = 300,000,000$

$b = 10$

→ b examples
→ 1 example
Vectorization

1.3: Comparision: Mini-bacth vs Batch, stochasic,

>1. Mini-bacth vs Batch
Mini-batch faster :
(Each iteration) start making progress in modifying the parameters after looking at just b examples rather than needing to wait till have scan through every single training example.

>2. **Mini-bacth vs Stochastic gradient descent**
Mini-batch outperform stochastic : only if have a good vectorized implementation

In each iteration, sum over b examples (for the derivation of J over paramter θ) in a more vectorized way will allow to partially parallelize computation over the b examples.
using appropriate vectorization to compute the derivatives, can sometimes partially use the good numerical algebra libraries and parallelize gradient computations over the b examples.
Whereas with Stochastic gradient descent , just looking at one example at a time is less to parallelize over.

>3, Disadvante of mini-batch:
Have one more parameter to faddle with: mini-batch size b. which may therefore take time.
But if have a good vectorized implementation this can sometimes run even faster that Stochastic gradient descent.

Summary:
Mini-batch gradient descent algorithm in some sense does something that's somewhat in between what Stochastic gradient descent and bactch gradient desent does.
And **if choose a reasonable value of b (usually use 10, , anywhere from say 2 to 100 would be reasonably common), and use a good vectorized implementation**, sometimes it can be faster than both Stochastic gradient descent and faster than Batch gradient descent.

18_135. Stochastic gradient descent convergence_Large scale machine learning

Purpose: Techniques: for making sure stochastic gradient descent algorithm is converging and for picking the learning rate a.

1. Checking for covergence

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

>1. Batch gradient descent:

Plot the optimization cost function J(θ) as a function of the number of iterations of gradient descent.
(note: plotting J_train-as not trained well yet).
J(θ)= 1/2 average of , sum: error square of prediction to label, over all m training example.

we would make sure that this cost function is decreasing on every iteration.

Problem:
When the training set sizes were small, we could do that because we could compute the sum pretty efficiently. But when have a massive training set size then you don't want to have to pause your algorithm periodically in order to compute this cost function since it requires a sum of entire training set size.
(batch norm: compute cost periodly, not on every iteration-too slow
stochastic : continue compute cost on every iteration/step/example, average over cost of last like 1000 example-**running estimate**)

Checking for convergence

- Batch gradient descent:
  - Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent.
  - $J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ M = 300, 000, 000
- Stochastic gradient descent: $(x^{(i)}, y^{(i)})$, $(x^{(i+1)}, y^{(i+1)})$, ...
  - $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$
  - 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 algorithm.

>2. Stochastic gradient descent
The whole point of stochastic gradient was that start to make progress after looking at just a single example without needing to occasionally scan through your entire training set right in the middle of the algorithm.

Check the algorithm is converging:

>1. Cost function used: cost (θ, x(i), y(i)) = 1/2 * (h -y) ^2
Cost of the parameters θ with respect to a single training example is just one half of the square error on that training example.

>2. During learning, compute cost (θ, x(i), y(i)) before updating θ using (x(i), y(i)).
While the algorithm is looking at the example x(i), y(i), but before it has updated the parameters θ using that an example,compute the cost of that example.

Do it in this way because: if we've just updated θ using one example, algorithm might be doing better on that example than on new example which would be representative.
cost computed with examples just used in last iteration, might be smaller than cost computed with new examples, as h(x) parameters just updated by fitting example used in last iteration.

>3. Every certain number (e.g 1000) iterations, plot cost (θ, x(i), y(i) averaged over the last certain number (1000) examples processed by the algorithm.
---> **gives a running estimate** of how well the algorithm is doing on the last 1000 training examples that your algorithm has seen. And by looking at those plots, this will allow us to check if stochastic gradient descent is converging.

So, in contrast to computing J_train periodically in Batch gradient descent which needed to scan through the entire training set. With above procedure, as part of stochastic gradient descent, it doesn't cost much to compute these costs as well right before updating to parameter theta. And all we're doing is every thousand iterations or so, we just average the last 1,000 costs that we computed and plot that. And by looking at those plots, this will allow us to check if stochastic gradient descent is converging.

---

2. Stochastic gradient descent cost plotting example

2.1 Plotting example1: different learning rate (for converged paramters)

>1. Cost averaged over 1000 examples:- blue curve:
cost function decreasing with noise, and finally flattened out-algorithm maybe has converged.

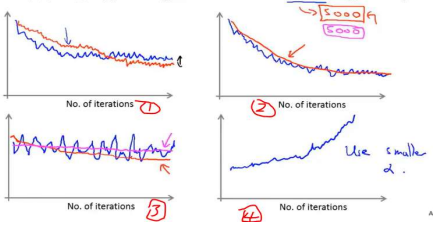
Noisy: cost ploting is a little bit noisy , due to it may not decrease on every single iteration. which is because cost is averaged over just a small subset (e.g 1000 training examples here)

Figure 1 -blue cureve looks like that would be a pretty decent run with the algorithm, where it looks like the cost has gone down and then starting from around one point/iteration. this plateau looks kind of flattened out, then maybe learning algorithm has converged.

>2. use smaller learing rate vs above algorithm- red curve
Algorithm may initially learn more slowly so the cost goes down more slowly.
And eventually actually algorithm is possible to end up at maybe very slightly better solution.

Checking for convergence

Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples



End up at maybe very slightly better solution:
Because stochastic gradient descent doesn't just converge to the global minimum, instead the **parameters will oscillate （震荡） a bit around the global minimum. So by using a smaller learning rate, you'll end up with smaller oscillations.**
And sometimes this little difference will be negligible and sometimes with a smaller learning rate can get a slightly better value for the parameters.

2.3 Plotting example3: different size of avearged examples (for unclear convergence paramters)

>1. Cost averaged over small (2/ 3/1000)examples:- blue curve;
Could not see error decrease and parameter converged or not with iteration increase, **very noisy.** Seems algorithm is not learning.

>2. Cost averaged over 5000 examples
increase averaging over large size example: 5000, go to below two cases:

>>1. Red curve -Looks like the cost actually is decreasing
It's just that the blue line averaging over 2, 3 examples, was too noisy so couldn't see the actual trend in the cost actually decreasing. and possibly averaging over larger, 5,000 examples instead of 1,000 may help.

>>2. Pink curve: still flat
when you average over a larger number of examples, error cuver still flat, then that's maybe just a more firm verification that unfortunately the algorithm just isn't learning much for whatever reason.

need to either change the learning rate or change the features or change something else about the algorithm

2.2 Plotting example 2: different size of avearged examples (for converged paramters)

(note: increase averaged example size, do not change optimization result-final parameter/cost result.
only change averaged cost during this optimizing process-->less noise-flatten out to the same final error value)

>1. Cost averaged over 1000 examples:- blue curve;

>2. Cost averaged over 5000 examples:-R:ed curve:
Error curver is smoother by averaging over larger, say 5,000 examples .
The disadvantage:
feedback get on how well your learning learning algorithm is doing is more delayed, because you get one data point on your plot only every 5,000 examples rather than every 1,000 examples.

2.4 Plotting example4: error curve increase with iteration increase
this is a sign that the algorithm is diverging. and should use a smaller value of the learning rate.

Summarize:
Above gives a sense of the range of phenomena you might see when you plotting these cost average over some range of examples as well as suggests the sorts of things might try to do in response to seeing different plots.
So if the plots looks too noisy, or if it wiggles up and down too much, then try increasing the number of examples is averaging over so can see the overall trend in the plot better.
And errors-cost are actually increasing, try using a smaller value of learning rate a.

(note:
smaller learning rage--->
1. J_train ploting smoother-better the J_train trend (decrease or not);
2: final coverge to parameter fitting training set better-J_train smaller (as oscillate in a smaller region of global optima)
3. gradient descent step is smaller-->parameter update slower-J_train decrease slower-need more time to train
-->J_train decrease slower, smoother, and finally end up with smaller value-better solution-parameter.
4. could used to fix un-converge problem)

larger J_train averaged example--->
no influence on paramter updating during iterations, no influence on each iteration cost error. **just method to show error info.** -->
1. J_train feedback delayed (as need to average on larger examples);
2. J_train plotting smoother-average cost ocsillation of each example on larger scale example
)

3. Learning rate:

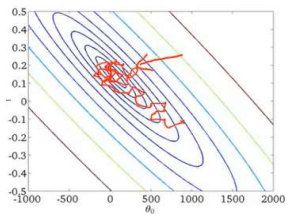
Stochastic gradient descent algorithm will start from initial point and sort of meander towards the minimum And then it won't really converge, and instead it'll wander around the minimum forever.
--> end up with a parameter value that is hopefully close to the global minimum that won't be exact at the global minimum.

>1 Learning rate is constant
In most typical implementations of stochastic gradient descent, the learning rate alpha is typically held constant. And so will end up is exactly a picture likeright-red curve, oscillating around global optima.
(note:
parameter updating step is actually smaller, as derivate is decreasing while close to global optima: a* derivative is decrease, parameter **θ:=θ- a* derivative i: parameter updating step is actually smaller)**

>2. Changing learing rate:
slowly decrease the learning rate over time/iteration, **would help tochastic gradient descent actually converge to the global minimum,**

Typical: α = c1 / (iteration number + c2)
iteration number: really the number of training examples alogirthm has seen
c, c2: additional parameters of the algorithm, in order to get good performance.

Keep learing rate more as constant, beacuse otherwise have 2 extra parameters, c1, c2 , makes algorithm more complicated.
But manage to tune these two parameters well, then algorithm will actually around towards the minimum, but as it gets closer and learning rate is decreasing, the meanderings will get smaller and smaller until it pretty much just to the global minimum.



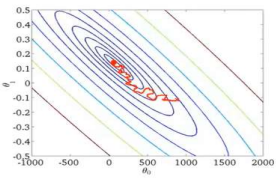
Stochastic gradient descent

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

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset.
2. Repeat {
   for $i := 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
   (for $j = 0, \ldots, n$)
   }
   }

Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (E.g. $\alpha = \frac{const1}{iterationNumber + const2}$)



Summarize:
If do slowly decrease alpha to zero you can end up with **a slightly better** hypothesis. But because of the extra work needed to fiddle with the constants and because frankly usually we're pretty happy with any parameter value that is pretty close to the global minimum.
Typically this process of decreasing slowly is usually not done and **keeping the learning rate constant is the more common application of stochastic gradient descent** although you will see people use either version.

note: derivative of gradient dθ is actully decrease while aheading to global optimum, more to 0. ->even learnign rate a is constent, gradient descent step- a * dθ is actually decreasing to 0.

Summary this item:
A way for **approximately monitoring how the stochastic gradient descent is doing in terms for optimizing the cost function**. And this method that does not require scanning over the entire training set **periodically** to compute the cost function on the entire training set. But instead it looks at say only the last thousand examples or so-**running estimation.**
And can use this method both to make sure the stochastic gradient descent is okay and is converging or to use it to tune the learning rate alpha.

18_136. Online learning__Large scale machine learning

Purpose:
model problems where we **have a continuous flood or a continuous stream of data coming in and we would like an algorithm to learn from that.**
e.g:
If you have a continuous stream of data generated by a continuous stream of users coming to your website, what you can do is sometimes use an online learning algorithm to learn user preferences from the stream of data and use that to optimize some of the decisions on your website.

1. Online learning example_shipping

1.1. Problem:
continuous coming users, checking delivery part price for their shipping: origian and destiantion;
-->create algorithm to optimize shipping price for user.
algorithm to predict if user will accept this price / choose this shipping service for their delivery after knowing the price.

1.2. Algorithm idea:
>1. Feature x: Capture user properties of the users: package delivery origin and destinaton + price offered for user shipping.

>2. Predict: probability user using one shipping service given these features (which also captures the price offered to user)" p(y=1|x, θ)
--> then can try to pick a price so that have a pretty high probability for user choosing this shipping company while simultaneously offering a fair profit for shipping user package.

(note:
classific problem?Z = W1T*x +b1, h =sigmoid (Z)? or h=sigmoid (W2*Z +b2)

1.3 . Model p(y=1|x, θ) using logistic regression
Note: could also use neural network instead of logistic regression.

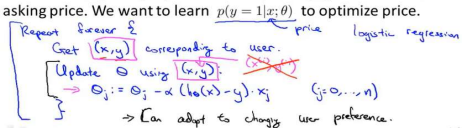
Assume website run continuously-having continuous online data:
---> Online learning algorithm peform:
> Repeat forever (website keep on staying up-contnious data): for below step 1-3.
>>1. get one example / user info. : (x, y)
Feature x: Capture user properties of the users: package delivery origin and destinaton + price offered for user shipping.
y=1, user choose this websit for shipping, =0 otherwise.

>> 2. Update parameter θ **based on above one example (x, y).**
θ:= θ -a * derivative (h-y)*x

>>3. Discard above example and never use again.
as use one example only once, and do not use x(i), y(i), only x , y here.

(note: for trained/being trained algorithm:-->get decision boundary: p(y=1|x,θ): f(x,θ) = c
θ is trained; one get new user/feature: destination , etc, choose price value , to make this new user feature vection meet decison boundary.
)

Online learning
Shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service ($y = 1$), sometimes not ($y = 0$).
Features x capture properties of user, of origin/destination and asking price. We want to learn $p(y = 1|x; \theta)$ to optimize price.



Summarize:

1. online learning algorithm can be very effective, if really have a continuous stream of data:
If have a continuous stream of users coming, then this sort of online learning algorithm is actually a pretty reasonable algorithm. Because the data is essentially unlimited then there is really may be no need to look at a training example more than once.
If had only a small number of users then might be better saving away all the data in a fixed training set and then running some algorithm over that training set.

2. online learning algorithm can adapt to changing user preferences:
if over time users preferance changed, online learning algorithm could keep tracking this change as it **adapt/update parameters θ to whatever latest pool of users data.**

2. On line learning example_learning to search

2.1. Problem:
Product searching: find certain numbers of product , return users in response to user-search query.

2.2. Algorithm idea:

>1. Feature vector x: created for each product and given specific user query
Capture product properties + similarity to user search query:
similarity to user search query:
how well the phone matches the user query along different dimensions, could check like how many words in the user search query match the name of the phone/ or match description of the phone and so on.

>2. Predict: probability that a user will click on the link for a specific product, p(y=1 | x, θ)
as the purpose is to show user product that they are likely to want to buy, then show the user product that they have high probability of clicking on in the web browser.
(note: application: then choose product /feature have high probability.)

y =1 if user clicks on the link; y = 0 otherwise
Learn p(y=1 | x, **θ**)
Learn the probability the user will click on a specific product given:
Features x, capture properties of the phone and how well the query matches the phone.

This learning problem is actually called: **predicted click-through rate, the predicted CTR.**
It just means learning the probability that the user will click on the specific link that you offer them.

>3. Predict CTR for any partifuclar product (e.g 100 products)

>4. Select the product have high CTR and show to user.
Can compute this for each of the 100 products and just select the 10 that the user is most likely to click on-pretty reasonable way to decide what ten results to show to the user.

Other online learning example:
Product search (learning to search)
User searches for "Android phone 1080p camera"
Have 100 phones in store. Will return 10 results.
→ x = features of phone, how many words in user query match name of phone, how many words in query match description of phone, etc.
→ $y = 1$ if user clicks on link. $y = 0$ otherwise.
→ Learn $p(y = 1|x; \theta)$. prediction CTR
→ Use to show user the 10 phones they're most likely to click on.
Other examples: Choosing special offers to show user; customized selection of news articles; product recommendation; ...

Summarize:

1. CTR algorithm will get certain number (like 10) training examples every time a user come
because for the ten product that we chose to show the user, we get a feature vector X and will also get a value for y, depending on whether or not user clicked on the products.

2. Could run websiste in this way:
>1. continuously show the user certain number best guesses for products they might like-->
>2. each time a user comes will get this certain number examples (x,y)pairs--->
>3. Use online learning algorithm to update the parameters using essentially 10 steps of gradient descent on these 10 examples (stochastic , not minibatch)
>4, Throw the data away

When have a continuous stream of users coming to website, this would be a pretty reasonable way to learn parameters for on line learning algorithm , so as to show the certain number products to users that may be most promising and the most likely to click on.

2.3 Other examples: that similar to phones
>1. Choosing special offer to show the user;
>2. Customized selection of news articles;.
>3. Product recommendation:;
if you have a collaborative filtering system, you can even imagine a collaborative filtering system giving you **additional features t**o feed into a **logistic regression classifier** to try to predict the click through rate for different products that you might recommend to a user.

(note:
training data: (x, y):
input: x: product feature (including similarity to search words)
lableled outpupt: y=1/0: click/no click

modle: p(y^|x, parameter): clicking probability given input product feature.

proccess: predict CTR for many products -->choose high CTR products , show to user-->get labeled y: click or not
-->get training examples, to update model parameter: use stochastic gradient descent.)

Note:
Any of these problems could also have been formulated as a standard machine learning problem, where you have a fixed training set.
Maybe, you can run your website for a few days and then save away a training set, a fixed training set, and run a learning algorithm on that. But where you do see large companies get so much data, that there's really maybe no need to save away a fixed training set, but instead you can use an online learning algorithm to just learn continuously from the data that users are generating on your website.

Summary: online learning algorithm

1. Very similar to this schotastic gradient descent algorithm, only instead of scanning through a **fixed training set,** we're instead getting one example from a user: learning from that example, then **discarding it and moving on.**

2. Advantage: can adapt learned hypothesis to whatever the latest sets of user behaviors are like as well.
if you have a changing pool of users, or if the things you're trying to predict are slowly changing like your user taste is slowly changing, the online learning algorithm can slowly adapt your learned hypothesis to whatever the latest sets of user behaviors are like as well.

If we have a continuous stream of data for some application, this sort of algorithm may be well worth considered.

18_137. Map-reduce and data parallelism__Large scale machine learning

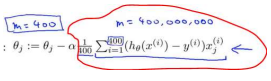
stochastic gradient descent, and other variations of the stochastic gradient descent algorithm, including those adaptations to online learning: all of those algorithms could be run on one machine, or could be run on one computer.
And some machine learning problems are just too big to run on one machine, sometimes maybe you have just so much data you just don't ever want to run all that data through a single computer, no matter what algorithm you would use on that computer.
Map reduce approach: a different approach to large scale machine learning, equal importance to stochastic, but simplier.
using these ideas you might be able to scale learning algorithms to even far larger problems than is possible using stochastic gradient descent.

1. Map-reduce_Example: Batch gradient descent

1.1 Batch gradient descent:
In each iteration, while computing derivation of cost function of parmater, need to sum over all training example.

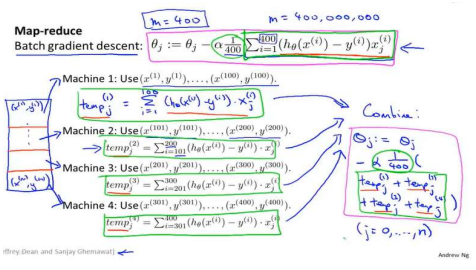
When m is large, computationly expensive-->**method : use many machines to compute this sum in parallel:**
Divide computation workload on different machine.

e.g: m=400,
>1. use 4 machines to compute derivation sum in parallely:
M1: compute first 100 example sum of derivation: temp1
...
M4: compute first 100 example sum of derivation: temp4



2. Combine all machine computation together
--->derivative = 1/m * (temp1 +temp2+ temp3 +temp4)

>3. update paramter: θ := θ- a/m * (temp1 +temp2+ temp3 +temp4)
>4. repeate step 1-step3.



1. 2. Map_reduce idea: divide derivation (suming over all example)workload into different machines.

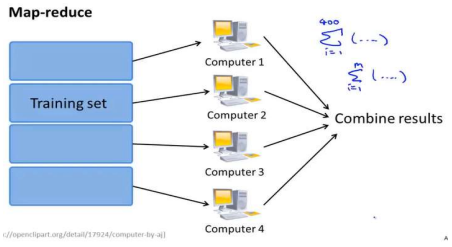
>1. split this training set equally into different subsets;

>2. Send to different computers for computing subset;

>3. combine all machines result in centralized server

e.g : if have 4 machines to run, potentially y get up to a 4x speed up.
if there were network latencies and costs of the network communications to send the data back and forth, then get slightly less than a 4x speedup.
(note: parameter used in each computer to computing sum, is same: h(θ, x(i)), only example x(i) is different.)

This approach offer a way to process much larger data sets than is possible using a single computer.



1.3. Map-reduce and summartion over the training set

Map Reduce : keep quesitoning if the learning algorithm can be expressed as a summation over the training set. (note: cost function/optimizing object & derivatives)
-->many learning algorithms can actually be expressed as computing sums of functions over the training set

e.g: for advanced optimization, with logistic regression, need J_train and derivative of J over parameter θ.

both J_train and derivative of J over parameter θ is sums over the training set-->could divide into subset examples for different machines, and combine together in central service, and get the overall partial derivative, which you can then pass through the advanced optimization algorithm.

Map-reduce and summation over the training set

Many learning algorithms can be expressed as computing sums of functions over the training set.

E.g. for advanced optimization, with logistic regression, need:

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

$$\frac{\partial}{\partial \theta_j} J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

1.4 Summarize:

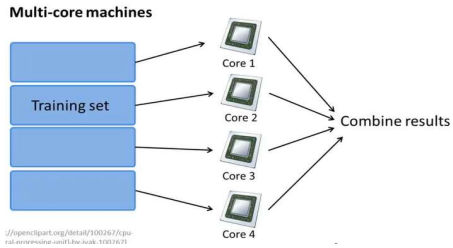
More broadly, **by taking other learning algorithms and expressing them in sort of summation form or by expressing them in terms of computing sums of functions over the training set**, you can use the MapReduce technique to parallelize other learning algorithms as well, and **scale** them to **very large training sets.**

2. Multi-cor machines

If you have just a single computer, MapReduce can also be applicable.

One compute can have multiple processing cores: you can have multiple CPUs, and within each CPU you can have multiple proc cores.
If you have a large training set, then can split the training sets into pieces and send the training set to different cores and use MapReduce this way to divide work load.
Each of the cores can then carry out the sum over, and then they can take the partial sums and combine them, in order to get the summation over the entire training set.

Advantage of parallizing in cores within a single machine, do not wory about network latency (网络延迟). because all the communication: all the sending of the temp result back and forth, happens within a single machine.



2.1 Summarize:
Depending on the details of your implementation, if you have a multi-core machine and if you have certain numerical linear algebra libraries, It turns out that the sum numerical linear algebra libraries that can automatically parallelize their linear algebra operations across multiple cores within the machine.
In that case, If have a very good vectorizing implementation of the learning algorithm, can just implement standard learning algorithm in a **vectorized fashion**. and do not worry about parallelization as numerical linear algebra libararies could take care of it. do not need to implement Map-reduce.

But for other any problems, taking advantage of map reducing commentation, finding and using this MapReduce formulism to paralelyz across course explicitly yourself might be a good idea as well and could let you speed up your learning algorithm.

3. Summary:

MapReduce approach is to parallelize machine learning by taking a data and spreading them across many computers in the data center. and these ideas are applicable to paralysing across multiple cores within a single computer as well.

Today there are some good open source implementations of MapReduce: Hadoop. and using either your own implementation or using someone else's open source implementation, you can use these ideas to parallelize learning algorithms and get them to run on much larger data sets than is possible using just a single machine.

W19 - Application example: Photo OCR

19_139. Roblem description and pipeline__Application example: Photo OCR

Purpose:
Talk a machine learning application history centered around an application called Photo OCR . To show:

1. how a complex machine learning system can be put together.
2. once told the concepts of a machine learning a type line and how to allocate resources when you're trying to decide what to do next.
And this can either be in the context of you working by yourself on the big application Or it can be the context of a team of developers trying to build a complex application together.
3. Photo OCR problem involves a couple more interesting ideas for machine learning:
   >1. One is some ideas: how to apply machine learning to computer vision problems,
   >2. Second is the idea of artificial data synthesis.

1. Photo OCR problem

we now have tons of visual pictures that we take all over the place. And one of the things that has interested many developers is how to get our computers to understand the content of these pictures a little bit better.
Phote OCR: Optical Character recognition （照片光学字符识别）.
The photo OCR problem focuses on how to **get computers to read the text to the purest in images that we take.**
it might be nice if a computer can read the text in this image so that if you're trying to look for this picture again you type in the words, and have it automatically pull up this picture.

1.1 Photo OCR problem procedure:
>1. Look through the image and detect text positon in image. (classfication-have text or not & localization)
>2. Look at these text regions and actually read the text in those regions.
It'll come up with these transcriptions of what is the text that appears in the image.

OCR, or optical character recognition of **scanned documents** is relatively easier problem, yet doing OCR from photographs today is still a very difficult machine learning problem.

The Photo OCR problem

1.2 Photo OCR applications:
Help our computers to understand the content of images better, could
>1. Helping blind people, eg. telling them the words that maybe on the street sign in front of them, with a camera that can look at what's in front of them.
>2. Car navigation systems: e.g car could read the street signs and help you navigate to your destination.
etc.

2. Photo OCR pipeline

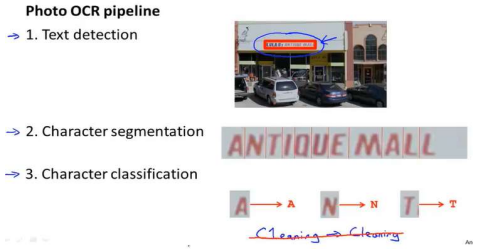
2.1 Perform photo OCR, steps:
1. Text detection:classfier1
First go through the image and find the regions where there's text and image.

>2. Character segmentation-classfire 2
Given the rectangle /box around that text region, then do character segmentation.
(note: machine translation /speech recognition using seq 2 seq model could not used here, as input could not be pre-processed to sequence word/input)
e.g: text box is "Antique Mall" and try to segment it out into the locations of the individual characters （一个个独立的字符区域）.
>3. Character classification:-classfire3
Run a classfier: looks at the images of the visual characters, and tries to figure out the each individual character: first an A, the second character's an N, and so on.

A system like this is called a **machine learning pipeline 机器学习流水线.**

(note: break down problems, and each probelm use a algorithm to fix)

Photo OCR pipeline



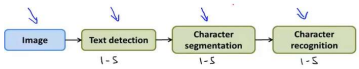
Note: some photo OCR systems do even more complex things, like a bit of spelling correction at the end: e.g: character segmentation and character classification system tells that it sees the word c l e a n i n g. Then sort of spelling correction system might tell this is probably the word 'cleaning', and character classification algorithm had just mistaken the l for a 1. -will not talk this here.

3. Photo OCR pipeline step pic.

Right is the picture showing the photo OCR pipeline.
have an image--->fed to the **text detection system** to find text regions ----> **segment out the characters**--the individual characters in the text ---->finally **recognize the individual characters.**

3.1 Common pipelines in machine learning:
>1. have **multiple modules**--text detection, character segmentation, character recognition modules-
>2. Each module may be machine learning component, or sometimes not.
Machine learning has a set of modules that act one after another on some piece of data, in order to produce the output. e.g: In photo OCR , is to output transation of photo text.

Photo OCR pipeline



3.2 Design machine learning system:
one of the most important decisions will often be **what exactly is the pipeline that you want to put together.**
e.g photo OCR problem, how to break this problem down into a sequence of different modules. The performance of each modules in pipeline, will often have a big impact on algorithm final performance.

Team work on pipeline: may divide workload amongst different members in a team.
May have different people work on different module seperately, e.g: 1 to 5 engineers for character segmentation, 1-5 engineers for character recognition, and so on.
while, all of this work could also be done by just one person if want.

Summary:
In complex machine learning systems the idea of a machine of a pipeline, is pretty pervasive （渗透）.
Above is a specific example of how a Photo OCR pipeline might work.

19_140. Sliding windows__Application example: Photo OCR

1. Tex detection

The first stage of the filter was the Text detection where we look at an image like this and try to find the regions of text that appear in this image.

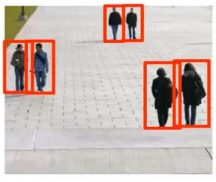
Text detection is an difficultl problem in computer vision. Because depending on the length of the text you're trying to find, these rectangles that you're trying to find can have different aspect.

->start with a simpler example of pedestrian detection, Ideas that were developed in pedestrian detection can apply to text detection.

(note: object classification + localtion for computer vision:
1. convolution implementation of sliding window-classfier output y=[pc, c1, c2, c3..]-object size not accuracy-only limited to classfire input size;
2. YOLO: put grid on image-->run LetNet/AGG16 etc. map input image to output: grid number x each grid labeled output yi =[pc, bx, by, bh, bw, c1, c2, c3...]--only one object in each grid;
3. Anchox box: based on YOLO, only for each grid , output yi=[yi_1; yi_2],yi_1, yi_2 associated with different bounding box-maching measured by IOU with predicted object size bw, bh.--only detect object with different shape.
text bounding box: shape different, not fixed bouning box to match-->could not use above method??)

Text detection

Pedestrian detection

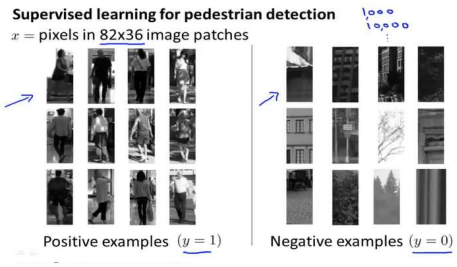


2. Example: Pedstrian detection
Ideas that were developed in pedestrian detection and apply them to text detection.

2.1 Object: find the individual pedestrians that appear in the image.

Ojbect box: rectangules ratio (length and width) is fixed in pedstrian example.
Simpler than text detection problem.
Because the aspect ratio of most pedestrians are pretty similar, the pedestrians can be different distances away from the camera and so the height of these rectangles can be different depending on how far away they are. but the aspect ratio is the same. so just using a fixed aspect ratio for these rectangles that we're trying to find.
But for text detection the height and width ratio is different for different lines of text.

Let's say that we decide to standardize on this aspect ratio of 82 by 36 and we could have chosen some rounded number like 80 by 40 or something, but 82 by 36 seems alright.

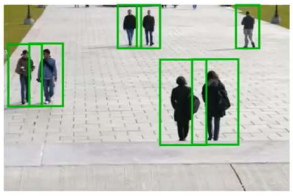


2.2 . Supevised learning for pedestrian detection

2.2.1 Process for classfying have pedestrian or not.-classification
e.g: setting aspect ratio = 82 x 36
> 1. Collecting large training sets: 82 X 36 image (classfier input size), typical size m is about 1,000 /10,000
Contain positive (have pedestrian)and negative examples (do not have pedestrian)

>2. Train algorithm on training set
To classify that input/image contain a pedestrian (y=1) or not (y=0).

This is applying supervised learning in order to take an imag and then can determine whether or not a pedestrian appears in that image capture.



2.3: Sliding window detection

e.g: pedstrian finding
Problem: Get a new image, try to find a pedestrian in the image.
Process:
>1. Take a certain size rectangular patch of this image.

>2. Repeat:
>>>2.1. run that image patch through classifier have trained in 3.1.
To determine whether or not there is a pedestrian in that image patch.
>>>2.2. slide rectangle over a bit, take next patch of this image, and run on classfier to predict have pedestrian or not.

Note:
Slide paramter/step size: the amount by which to shift the rectangle over each time is a parameter, t you step this one pixel at a time. step small usually performs best, but more computation expensive.
(note: image patch rescaled to 82x46: input size of classfire: -->object box will the times of classfire's input size: 82x46)

>>>2.3 .Repeat step 2.1, 2.2until slide this window over the different locations in the image.

After this, get the detection pedestrian for one specific size-sliding window size.

>3. Take a different size rectangular, run step2.
Run all this size images patches through classsfier. repeate step 2.

Note:
when take a larger rectangular, means larger image patch, need resize it down to classfier size (like 82x36 axile): re-size it to be smaller image and then pass it through classifier to try and decide if there is a pedestrian in that patch.

>4. Can use an even larger scales and run step2.

Summary :
After this whole process: train a supervised learning classifier, and then use a sliding windows classifier/ detector of different size in order to find pedestrians in the image.
(actually is : convolution implementation of sliding window-trained classfier, run with patches covered by ~~different sliding window size~~ classfire's input size, on all image position)

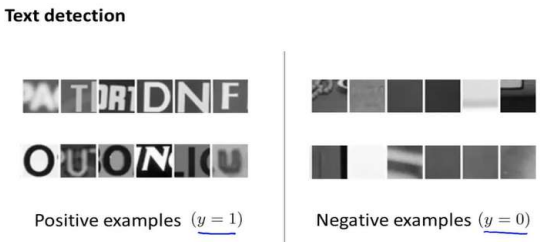
3. Text detection:

Gol: find the text regions in image.

3.1: Supervised leanring classfier (for text)
train a algorithm/ classfier on positive examples (have character in image) and negative examples (no character in image), to detect whether there is character in image or not.

>3.2 Test classfier on new image:
use one fixed scale sliding window (**one size window**) , run classfier on all image patches .

Result:
White area means text detection system has found text (axises of original image and result image is same).
Note: color change in result image: from back-->grey-->white, corresponding to the **probability of classfier detected there is text in that image position**.



As the goal is to draw rectangles around all the region where this text in the image, so we were going to take one more step which is we take the output of the classifier and apply to it what is called an expansion operator.

>3.3: Apply expansion operator
Expansion operator Effect: takes each of the white blob/regions and expands /grow that white region, by coloring those nearby pixels of white blob in white as well.

Mathematical way: create a new image -right pic, based on result image-left:
for every pixel in result image, check if it is whithin some distance of a white pixel. If a specific pixel is within **five pixels or ten pixels** of a white pixel, then we'll also color that pixel white in the new created image-right.
(how to define the distance, common 5-10 pixel distance?)

>3.4 Draw bounding around text
In the new created image:

>> 1. Rule out rectangles that have funny aspect ratios.
E.g: text bounding should be much wider than tall, so ignore the other white blobs that have funny (thin, tall) ratio bounding.

>>2. look at the contiguous (相邻的) white regions and draw bounding boxes around them.

Note: in this examples, missed to recognize text writen on window, which is hard to read.
(note: text detection:
1. method here used is:
>> Sliding window-**only one window size**: trained classfier for text detection first , then run on image patches using sliding window-text size not accurate even use multi window size, while here only use one window size.-could use convolution implementation of sliding window for computation cost saving.
>> text bounding box size: expansion operator: could same method used for other computer vision problem?
YOLO /anchox could not use here?-multi objects, more than two objects will occure in one grid??)

4. Character segmentation_1D sliding window

Having found these rectangles with the text in it, we can now just cut out these image regions and use pipeline to try to read the texts.

>4.1 Supervised learning classfier
Classfier function: look at the image patch and try to **decide if there is split between two characters right in the middle of that image patch.**

Train classfier with postive examples and negative examples:
positive example: could split two characters by putting a line right down the image middle; middle of positive examples represents a gap or a split of two characters.
Negative example: middle of negative examples do not represents a gap or a split of two characters.

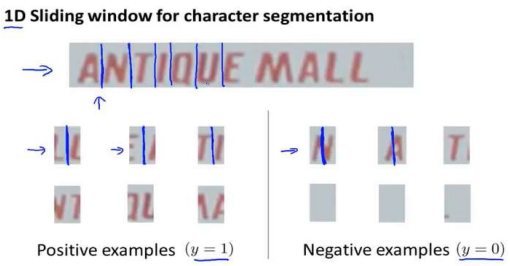
>4.2: Run classfier on text image (text dection system has pulled out)

>>1. Take a specific size sliding wondow (same size with classfier)

>>2. Take image patches by sliding window and run classfier on it: check if there is split in the middle of image patch for two characters. when classified y=1, could then draw a line in the image patch middle.
Note: sliding window here is 1D, as only sliding window over straight line/ only one row.

>>3. Run classfier in all postions of image
-->get right positon to split image into individual characters.

(Note: could also run convolution implementation of slidng window)



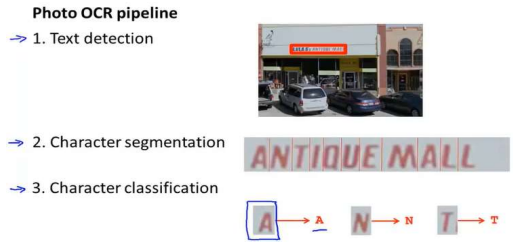
5.Photo OCR pipeline_Overall

>1. text detection step:
Run trained supervised learning classfier , with sliding windows to detect text in image (classifiy and localize)

>2. character segmentation
Run trained supervised learning classifer, with one-dimensional sliding windows to do character segmentation, segment this text image into individual characters. (classifiy and localize)

> 3. character classficiation
Use supervised learning muliti classfier, take image contained a character and decide what is the character that appears. (softmax: activation function)

Summary:
Photo OCR pipeline **use ideas like sliding windows classifiers** in order to put these different components to develop a photo OCR system.



19_141. Getting lots of data: Artificial data synthesis__Application example: Photo OCR

One of the most reliable ways to get a high performance machine learning system is to take a low bias learning algorithm and to train it on a massive training set.

Artificial data synthesis, this doesn't apply to every single problem, and to apply to a specific problem, often takes some thought and innovation and insight.
But if this idea applies to your machine, only problem, it can sometimes be an easy way to get a huge training set to give to your learning algorithm.

Artificial data synthesis comprises of two variations:
1. Creating new data from scratch
2. Already have a small label training set and then amplify that training set or use a small training set to turn that into a larger training set.

1. Example_Character recognition
Classfier goal: take an image patch and recognize the character in the middle of that image patch.

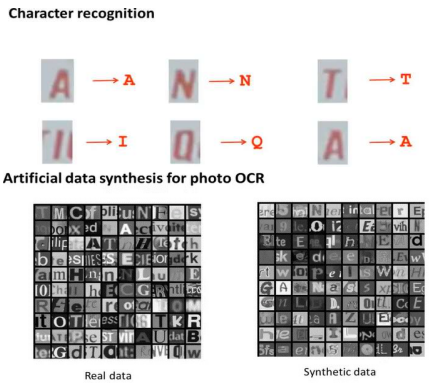
training set: real data + synthetic data

1.1. Real data
use grey image as training set, as color image does not help that much for character recoginition problem

1.2. Synthetic data_creating from scratch
>1. Take characters (may download from web)from different fonts, and **paste them agains different/random background.**
>2. Could **apply a little blurring operators,** affine distortions （仿射变换: sharing-等分， scaling, rotation, etc.),

-->this could created unlimited labeled data for supervised learning problem.

Note:
it takes thought at work, in order to **make the synthetic data look realistic**, and if you do a sloppy job in terms of how create the synthetic data then it actually won't work well.

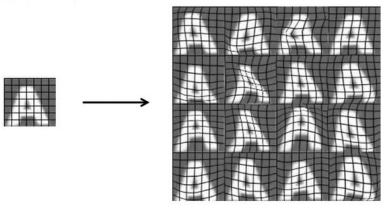


1.3 Synthetic data_amplify real data

Already have a small label training set and then creating extra data to amplify training set.
Right example: real data A have grids which added here only for illustration synthetic data.

Method
> 1. take real example image, and **introduce artificial distortions**-->create mulitiple new examples

Again, in order to do this for application, it does take thought and insight to figure out:
what **reasonable sets of distortions, or what are reasonable ways to amplify and multiply training set.**
For the specific example of character recognition, introducing these warping seems like a natural choice, but for a different learning machine application, there may be different the distortions that might make more sense.

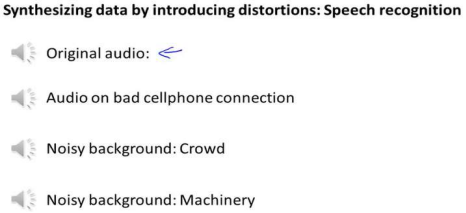


2. Example_Speech recognition_synthesizing data by introducing distortions
Goal: recognize words spoken in that audio.

Real data: a labeled example
-->amplify data to train algorithm.

synthesizing data:
>1. Intro. Additional audio distortionsinto data set.--->multiple examples
add noise: walking noise, train noise...etc.

--> By introducing different background sounds, we've now multiplied this one example into many more examples.
without much work by just automatically adding these different background sounds to the clean audio.

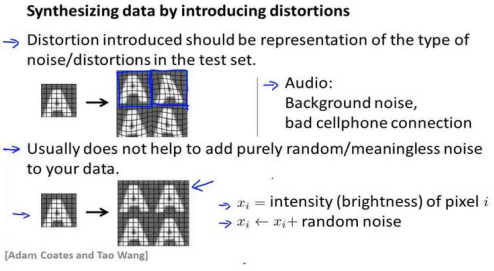


3. Synthesizing data by introducing distortions

Attention:
>1. Distortions introduced should be representative of the type of noises / distortions (you might see) in the test set.

for the character recognition example, warping introduced are actually kind of reasonable, beacause created new examples could be an image that we could actually see in a test set.
And for audio, do wanna recognize speech, even against a bad cellphone connection, against different types of background noise. so we're again **synthesizing examples are actually representative of the sorts of examples that we want to recognize correctly.**

>2. Usually it does not help to add purely random/meaningliess noise to your data.
e.g: taken a real image, and for each pixel in each of these 4 images, added some random Gaussian noise-pixel brightness. this is just a totally meaningless noise, unless you're expecting to see these sorts of pixel wise noise in your test set.



Summarize:
The process of artificial data **synthesis is a little bit of an art and sometimes you just have to try it and see if it works.**
But if you're trying to decide what sorts of distortions to add, do think about what are the **meaningful distortions** you might add that will cause you to generate additional training examples that are at least somewhat **representative of the sorts of images you expect to see in your test sets.**

4. Discussion on getting more data

4.1. Make sure classfier is low bias, before expending a lot of effort, figuring out how to create artificial training. only for a low bias classfier, large data will really help.

>> 1. Plot the learning curves J_train and J_dev, and make sure only have a low bias or high variance classfier.
otherwise keep increasing the number of features /number of hidden units in neural network, until have a low bias classfier.
so as to avoid spending long time to get a great artificially synthesized data set and only to realize afterward, that learning algorithm performance doesn't improve that much, even when you're given a huge training set.

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)

4. 2. check: 'how much work would it be to get 10 times as much date as we currently have?'

Good question to ask ourself or team, as it's really not that hard, maybe a few days of work at most, to **get ten times as much data** as we currently have for a machine running application.
Data collecting method:
>1. Artificial data synthesis:
generating data from scratch or taking an existing example and introducing distortions that amplify to enlarge the training set
>2. Collect/label data by yourself.
estimate how long takes to label: normally not much time (days/few weeks) to label by oneself to get 10 times of current data-->great way to get much better performance.
>3. 'crowd sourcing ' (E.g Amazon Mechanical Tuk)
Today, there are a few websites or a few services that allow you to hire people on the web to, fairly inexpensively label large training sets for you.

Summary: note: low bias algorithm + huge data (artificial synthesis or label more data per se)
Artificial data synthesis idea, either creating new data from scratch or by taking existing label examples and introducing distortions to it, to s**ort of create extra label examples.**

If facing a machine learning problem, it is often worth doing two things. One is sanity **check with learning curves, that having more data would help**. And second, **ask yourself seriously: what would it take to get ten times as much creative data** as you currently have, and not always, but sometimes, you may be surprised by how easy that turns out to be, maybe a few days, a few weeks, and that can be a great way to give your learning algorithm a huge boost in performance.

19_142. Ceiling analysis: What part of the pipeline to work on next__Application example: Photo OCR

One of the most valuable resources is developer time. there are cases spending a lot of time working on some component and only realize later that all that worked just doesn't make a huge difference on the performance of the final system.
When working on the pipeline machine on your system, this can sometimes give you a very strong signal / guidance on what parts of the pipeline might be the best use of your time to work on.

1. Estimating the errors due to each component (ceiling analysis)

Example: photo OCR problem.

1.1 Check what part of the pipeline should spend the most time trying to improve:
e.g find which one of the 3modules: Text detection, character segmentation, character recogniton.

1.2 Ceiling analysis process:
>1. Run trained algorithm on dev /test set-->get evaluation value:v1

>2. Choose one module: monkey around with the dev/test set
e.g: choose text detection, **for evey dev/test example, provide correcte text detection outputs:** tell algorithm where the text is in each of the dev/text example.
-->simulate what happens if have a text detection system with a 100% accuracy: disable text detection algorithm, instead manully label text location in dev/test set

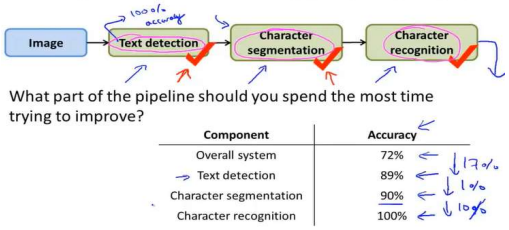
>3. Feed above image to rest of pipeline, **use same evaluation metrix** as before, and get evaluation value: v2

>4. Go to next module: do as step2: disable this module algorithm, just give the correct output manually labeled to next stage and run rest modul algorithm, get evaluation value3.

>5. Repeate until go to the last module, and get 100% accuracy.
Note: the evaluation value get in every step is : 1. for the overall system; 2. on dev set.

(note: -->dev/test need to be labeled in all moduels' output not only for the final output)

Estimating the errors due to each component (ceiling analysis)

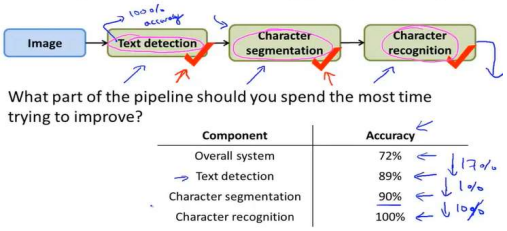


1.3: Ceiling analysis idea:
To understand what is the **potential upside for improving each of these components**, or

Compare evaluation values of algorithm with disabled modul get in step 1.2, with evaluation values of algorithm with disabled previous modul.
Get an understanding: how much could algorithm performance could improve if working on that modul.
e.g:
>1. text detection disabled-->improve: 89%-72% = 17%
if working on improving text detection, could potentially improve system's performance by 17%.
>2. Character segmentation disabled--> improve: 90% -89% = 1%
-->even have perfect character segmentation, you performance goes up by only one percent.
>3. Character recogintion: 100% -90% =10%
if working on character recognition, could may improve system performance by 10%.

-->Understand:
how much could possibly gain if one of these components became absolutely perfect. And this really places an **upper bound on the performance of that system.**

Estimating the errors due to each component (ceiling analysis)



2. Example_Face detection
Goal: try to recognize the person shown in this image.

2.1 Face detection pipeline

>0. Camera image;

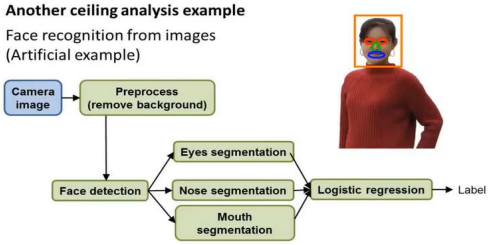
>1. Image pre-processing (e.g remove the background)

>2. Face detection (with learning algorithm)
Using sliding window classfier, draw box around the person face

>3 Face feature detection:
>>> Eyses segmentation (run classfier on face image)
>>> Nose segmentation (run another classfier on face image)
>>> Mouth segmentation (run another classfier on face image)

>4. logistic regression (with feeding face feature--->output label of the person)

Another ceiling analysis example
Face recognition from images
(Artificial example)



2.2: Ceiling analysis
Same as photo OCR ceiling analysis, step through each pieces module at a time:

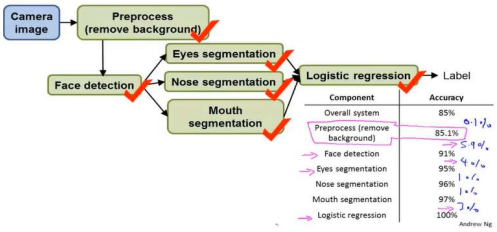
>0: run algorithm, get evaluation v1;

>1. Image pre-processing-->disable module function-manully remvoe backround and feed to rest pipeline, get system evaluation v2

>2. Face detection (with learning algorithm)
-->disable module function-manully label face and feed to rest pipeline, get system evaluation v3
>3 Face feature detection:
>>> Eyses segmentation
-->disable module function-manully label eye only and feed to rest pipeline, get system evaluation v4
>>> Nose segmentation
-->disable module function-manully label nose only and feed to rest pipeline, get system evaluation v5
>>> Mouth segmentation
-->disable module function-manully label mouth only and feed to rest pipeline, get system evaluation v6

>4. logistic regression (with feeding face feature--->output label of the person)
-->disable module function-manully label the person on image, get system evaluation v7

Another ceiling analysis example



---->as go through the system and just give more and more components the correct labels in the test set, the performance of the overall system goes up and you can look at how much the performance went up on different steps.

Note: there are people spend long time on image preprocess-backgound removal, while finally found algorithm system performance no much improval.

Summary:
Pipelines are pretty pervasive(普遍) and complex machine learning applications. And when you're working on a big machine learning application, don't waste your time working on something that ultimately isn't going to matter.

And the idea of ceiling analysis,is a very good tool for **identifying the component that you actually should put focus on** and make a big difference on the overall performance of your final system.

when have a sort of machine learning problem, do not trust your own gut feeling about what components to work on, instead, try the possibility to structure things and do **a ceiling analysis**. this is often a much better and much more reliable way for deciding where to put a focused effort, to really improve the performance of some component.
And be kind of reassured that, it will actually have a huge effect on the final performance of the overall system.
(note:
1. for one algorithm/module: do error analaysis -->direction to focus imporve that algorithm/model performance
2. for pipeline: -->ceiling analysis -->find module to work on)

为什么有人说 Python 的多线程是鸡肋呢?

在介绍Python中的线程之前，先明确一个问题，Python中的多线程是**假的多线程**！ 为什么这么说，我们先明确一个概念，全局解释器锁（GIL）。

Python代码的执行由Python虚拟机（解释器）来控制。Python在设计之初就考虑要在主循环中，同时只有一个线程在执行，就像单CPU的系统中运行多个进程那样，内存中可以存放多个程序，但任意时刻，只有一个程序在CPU中运行。同样地，虽然Python解释器可以运行多个线程，只有一个线程在解释器中运行。

对Python虚拟机的访问由全局解释器锁（GIL）来控制，正是这个锁能保证同时只有一个线程在运行。在多线程环境中，Python虚拟机按照以下方式执行。

1.设置GIL。

2.切换到一个线程去执行。

3.运行。

4.把线程设置为睡眠状态。

5.解锁GIL。

6.再次重复以上步骤。

对所有面向I/O的（会调用内建的操作系统C代码的）程序来说，GIL会在这个I/O调用之前被释放，以允许其他线程在这个线程等待I/O的时候运行。如果某线程并未使用很多I/O操作，它会在自己的时间片内一直占用处理器和GIL。也就是说，I/O密集型的Python程序比计算密集型的Python程序更能充分利用多线程的好处。

我们都知道，比方我有一个4核的CPU，那么这样一来，在单位时间内每个核只能跑一个线程，然后时间片轮转切换。但是Python不一样，它不管你有几个核，单位时间多个核只能跑一个线程，然后时间片轮转。看起来很不可思议？但是这就是GIL搞的鬼。任何Python线程执行前，必须先获得GIL锁，然后，每执行100条字节码，解释器就自动释放GIL锁，让别的线程有机会执行。这个GIL全局锁实际上把所有线程的执行代码都给上了锁，所以，多线程在Python中只能交替执行，即使100个线程跑在100核CPU上，也只能用到1个核。通常我们用的解释器是官方实现的CPython，要真正利用多核，除非重写一个不带GIL的解释器。

我们不妨做个试验：

难道就如此？我们没有办法在Python中利用多核？当然可以！刚才的多进程算是一种解决方案，还有一种就是调用C语言的链接库。对所有面向I/O的（会调用内建的操作系统C代码的）程序来说，GIL会在这个I/O调用之前被释放，以允许其他线程在这个线程等待I/O的时候运行。我们可以把一些 计算密集型任务用C语言编写，然后把.so链接库内容加载到Python中，因为执行C代码，GIL锁会释放，这样一来，就可以做到每个核都跑一个线程的目的！

可能有的小伙伴不太理解什么是计算密集型任务，什么是I/O密集型任务？

计算密集型任务的特点是要进行大量的计算，消耗CPU资源，比如计算圆周率、对视频进行高清解码等等，全靠CPU的运算能力。这种计算密集型任务虽然也可以用多任务完成，但是任务越多，花在任务切换的时间就越多，CPU执行任务的效率就越低，所以，要最高效地利用CPU，计算密集型任务同时进行的数量应当等于CPU的核心数。

计算密集型任务由于主要消耗CPU资源，因此，代码运行效率至关重要。Python这样的脚本语言运行效率很低，完全不适合计算密集型任务。对于计算密集型任务，最好用C语言编写。

第二种任务的类型是IO密集型，涉及到网络、磁盘IO的任务都是IO密集型任务，这类任务的特点是CPU消耗很少，任务的大部分时间都在等待IO操作完成（因为IO的速度远远低于CPU和内存的速度）。对于IO密集型任务，任务越多，CPU效率越高，但也有一个限度。常见的大部分任务都是IO密集型任务，比如Web应用。

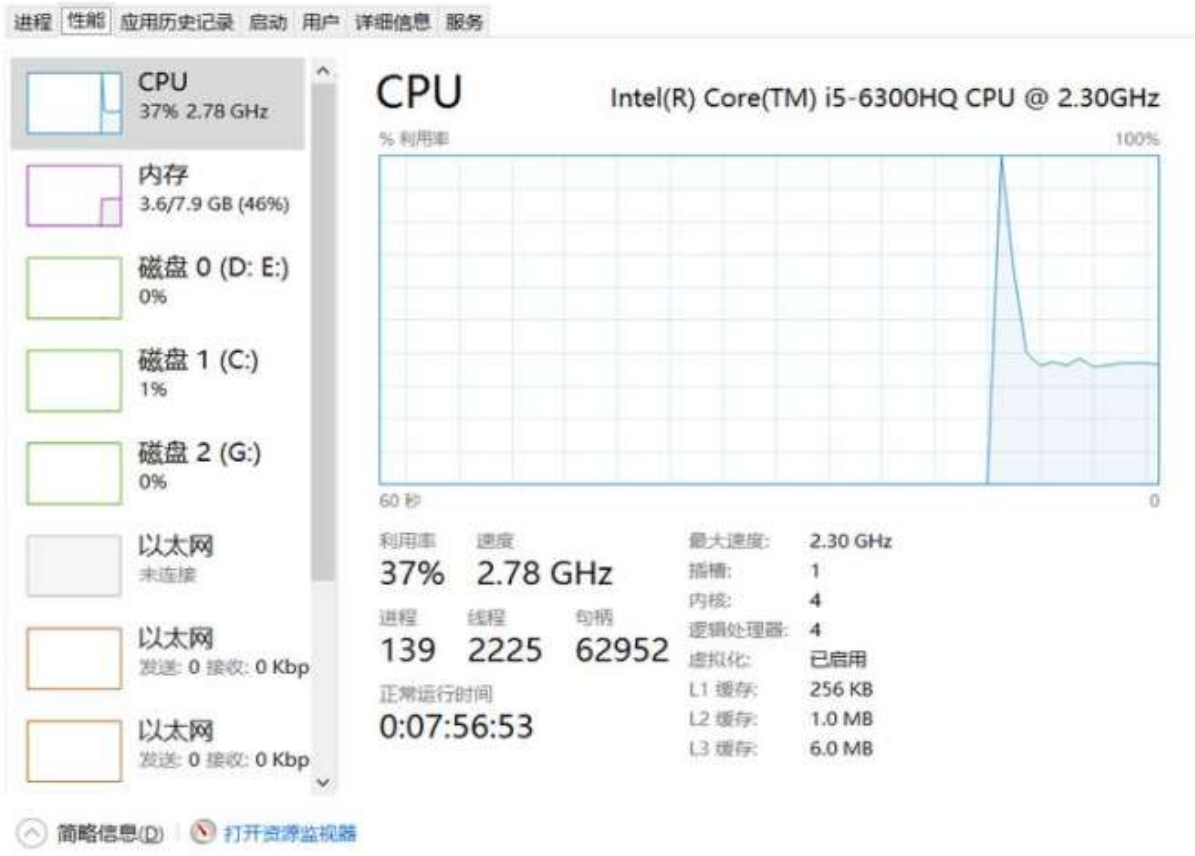
IO密集型任务执行期间，99%的时间都花在IO上，花在CPU上的时间很少，因此，用运行速度极快的C语言替换用Python这样运行速度极低的脚本语言，完全无法提升运行效率。对于IO密集型任务，最合适的语言就是开发效率最高（代码量最少）的语言，脚本语言是首选，C语言最差。

综上，Python多线程相当于单核多线程，多线程有两个好处：CPU并行，IO并行，单核多线程相当于自断一臂。所以，在Python中，可以使用多线程，但不要指望能有效利用多核。如果一定要通过多线程利用多核，那只能通过C扩展来实现，不过这样就失去了Python简单易用的特点。不过，也不用过于担心，Python虽然不能利用多线程实现多核任务，但可以通过多进程实现多核任务。多个Python进程有各自独立的GIL锁，互不影响。

我们不妨做个试验：

```
#coding=utf-8
from multiprocessing import Pool
from threading import Thread

from multiprocessing import Process


def loop():
    while True:
        pass

if __name__ == '__main__':

    for i in range(3):
        t = Thread(target=loop)
        t.start()

    while True:
        pass
```

我的电脑是4核，所以我开了4个线程，看一下CPU资源占有率：

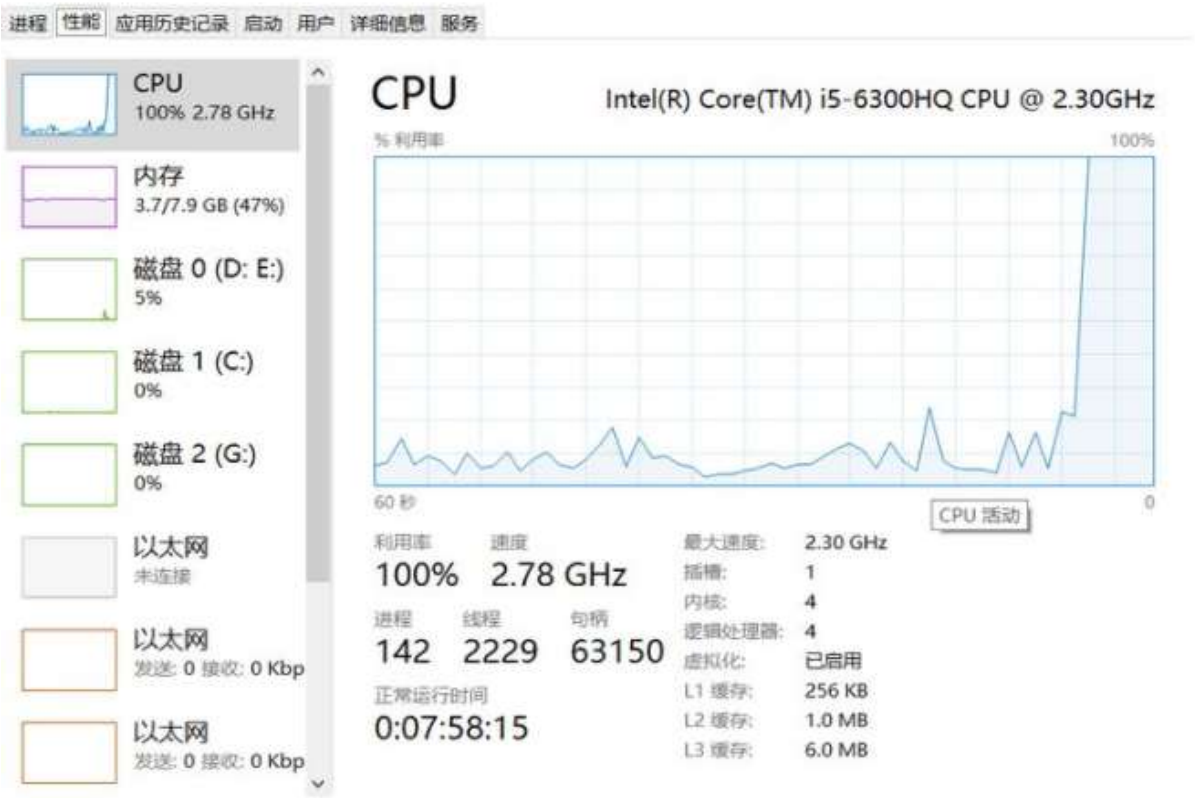


我们发现CPU利用率并没有占满，大致相当于单核水平。

而如果我们变成进程呢?

我们改一下代码:

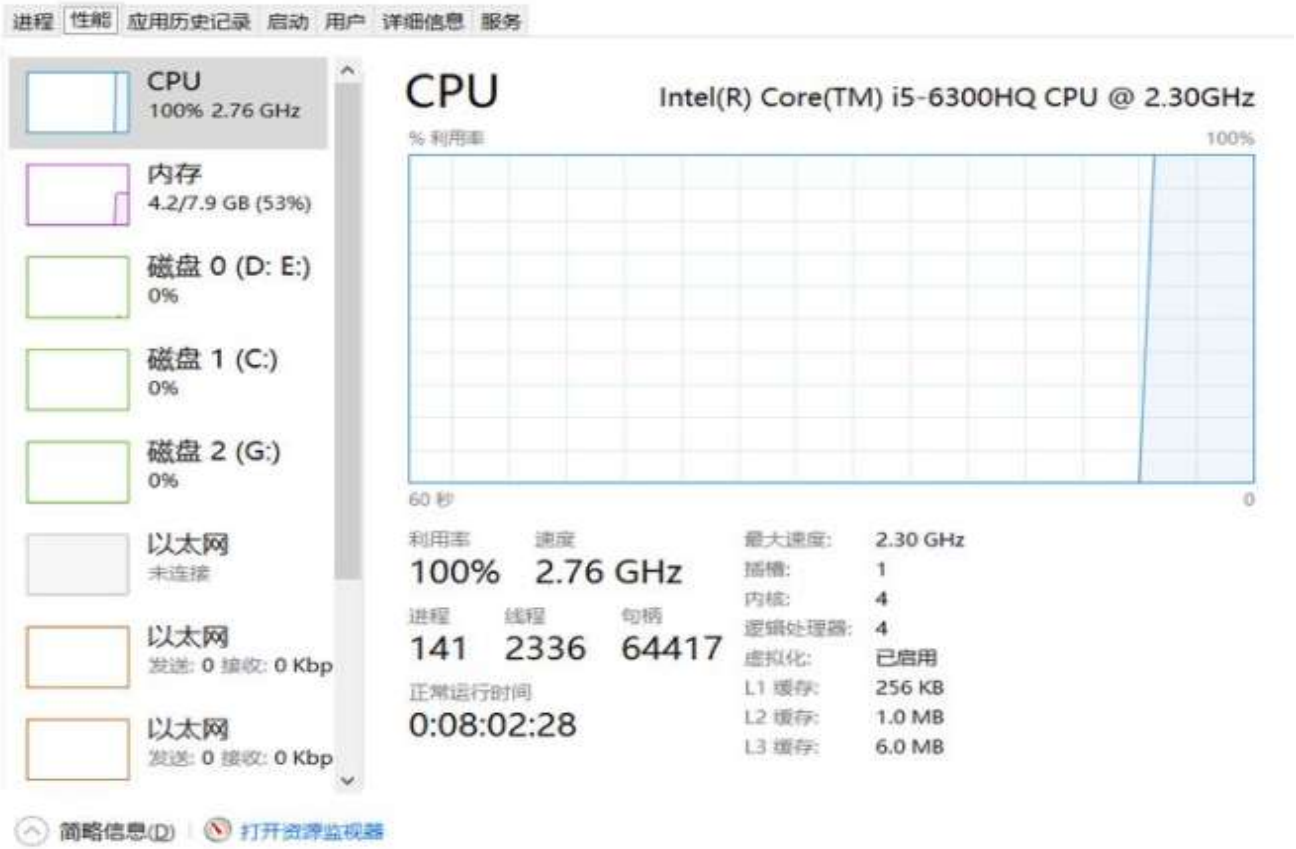
```
#coding=utf-8
from multiprocessing import Pool
from threading import Thread

from multiprocessing import Process


def loop():
    while True:
        pass

if __name__ == '__main__':

    for i in range(3):
        t = Process(target=loop)
        t.start()

    while True:
        pass
```



结果直接飙到了100%，说明进程是可以利用多核的!

为了验证这是Python中的GIL搞得鬼，我试着用Java写相同的代码，开启线程，我们观察一下：

```
package com.darrenchan.thread;

public class TestThread {
    public static void main(String[] args) {
        for (int i = 0; i < 3; i++) {
            new Thread(new Runnable() {

                @Override
                public void run() {
                    while (true) {

                    }
                }
            }).start();
        }
        while(true){

        }
    }
}
```



由此可见，Java中的多线程是可以利用多核的，这是真正的多线程！而Python中的多线程只能利用单核，这是假的多线程！