第一周 - Andrew Ng.

2017年7月24日　23:20

Machine learning: grow out of work in AI, have new capacity for computer

Range
- image recognition
- recommend system
- Database mining
- Autonomous helicopter
- Handwriting recognition
- Natural language processing
- Self-customizing program: AI

Definition
- E experience: collect what you want to know
- T task: based on E to do sth
- P probability: the number of correctly classified result

ML algorithms
- Supervised learning
- Unsupervised learning
- Reinforcement learning
- Recommender system

Fitting function
- Straight line
- Quadratic function, namely second-order polynomial

Supervised learning: right answers given
Unsupervised learning: predict continuous value output

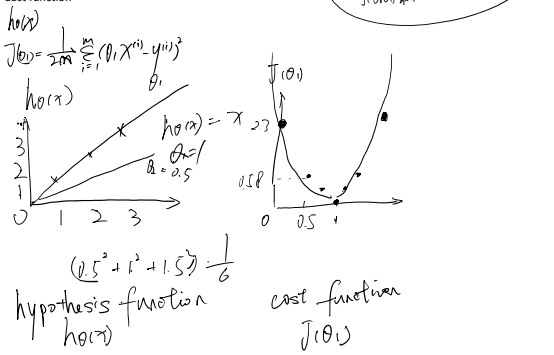
Classification: A or B: discrete value output (0/1) which several features/attributes impact them
Example
- regression: face recognition
- Classification: malignant or benign tumor

Unsupervised learning: approach problems with no idea of the results
Application: organize computer cluster, social network analysis (cohesive people), market segmentation, astronomical data
Octave function: SVD奇异值分解
e.g. SVD (singular value decomposition): linear algebra routine
Clustering application: different genes
Unclustering application: cocktail party algorithm: find structure in a chaotic environment

Model representation
1. Training set - learning algorithm - hypothesis

Cost function
- Squared error function is commonly used for linear regression

Hypothesis function
Cost function



$\frac{1}{6}(1.5^2 + 1^2 + 1.5^2)$

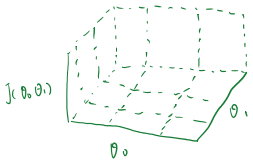
hypothesis function $h_\theta(x)$　　cost function $J(\theta_1)$

Find a θ which make J(θ) minimize, in this problem: when θ=1, have J(θ)min=0

算法的目标就是让代价函数最小化

高斯的对数似然与最小二乘
用来得到J(θ)

Gradient Descent Algorithm 梯度下降
Apply for linear regression



:= A denote assignment operator

Assignment $a := b$　　True assertion $a = b$

Repeat until convergence:

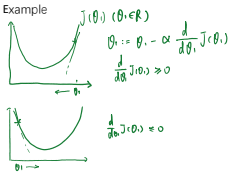
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

learning rate　微分项 derivative term

Simultaneous update-同步更新 相当于 gradient descent-梯度下降

$temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
$temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
$\theta_0 := temp0$
$\theta_1 := temp1$

Example



The learning rate: α
If α is too large, gradient descent can overshoot the minimun. It may fail to converge, or even diverge.

As we approach a local minimum, gradient descent will automatically take smaller steps (due to the decreased slope). So, no need to decrease α over time.

Gradient Descent for Linear Regression

gradient descent algorithm　　linear regression model

repeat until convergence:　　$h_\theta(x) = \theta_0 + \theta_1 x$

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$　　$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

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

$= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$

$= \frac{\partial}{\partial \theta_0} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$

$\Rightarrow$ 当 $j=0$ 时: $\frac{\partial}{\partial \theta_0} J(\theta_0,\theta_1) = \frac{\partial}{\partial \theta_0} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^{m} \left[\theta_0^2 + 2\theta_0\theta_1 x^{(i)} - y^{(i)} + (\theta_1 x^{(i)} - y^{(i)})^2 \right]$

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

当 $j=1$ 时: $\frac{\partial}{\partial \theta_1} J(\theta_0,\theta_1) = \frac{\partial}{\partial \theta_1} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})$

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

$= \frac{\partial}{\partial \theta_1} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$

$= \frac{1}{2m} \sum_{i=1}^{m} \frac{\partial}{\partial \theta_1} (\theta_0^2 + \theta_1^2 x^{(i)2} + 2\theta_0\theta_1 x^{(i)} - 2\theta_0 y^{(i)} - 2\theta_1 x^{(i)} y^{(i)} + y^{(i)2})$

$= \frac{1}{2m} \sum_{i=1}^{m} (2\theta_1 x^{(i)2} + 2\theta_0 x^{(i)} - 2x^{(i)} y^{(i)})$

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

repeat until converge

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

$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

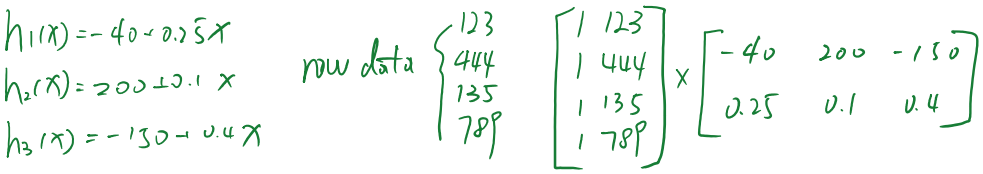
*Update theta 0 and 1 simultaneously

The cost function of linear regression is called convex function. ≈ bowl-shaped function
Character: not have local optima, but only one global optimum!

Linear Algebra Review

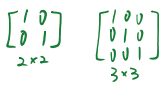
2017年8月12日 0:25

Scalar means that an object is a single value, not a vector or matrix. 标量
It is a real number.
Scalar multiplication 标量乘法
Vector: n by 1 or called n-dimensional vector



Associative property: YES - A*(B*C)=(A*B)*C
Commutative property: NO - A*B != B*A

Identity matrix: denoted I



For any matrix A, we have A*I = I*A = A
Matrix inverse:
If A is an m*m matrix (square matrix), and if it has an inverse.

Matrices that don't have an inverse called singular 奇异矩阵 or degenerate 退化矩阵
e.g. 

Matrix Transpose
Let A be an m*n matrix, and let $B = A^T$.
Then B is an n*m matrix, and $B_{ij} = A_{ji}$

第二周

2017年10月5日 10:54

Multivariate linear regression

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

For convenience of notation, define x0=1

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

$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$ ($x_0 = 1$)

$= \theta^T \cdot x$

Based on inner product of vectors, we have
theta transfer as theta transpose

$\theta^T = [\theta_0 \cdots \theta_n] \cdot x \Rightarrow \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix}$

m = the number of training examples
n = the number of features

Cost function

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

$\Updownarrow$

$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\sum_{j=0}^{n} \theta_j x_j^{(i)} \right) - y^{(i)} \right)^2$

Gradient Descent

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

$\Downarrow$ (Simultaneously update for all j)

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

(1)

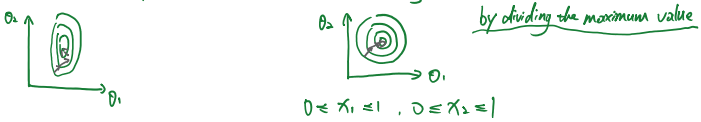
Feature scaling 特征缩放

idea: make sure features are on a similar scale, then gradient descent will converge more quickly.

x_1 = size (0 ~ 2000 feets) $\Rightarrow x_1 = \frac{size}{2000}$

x_2 = number of bedrooms (1 ~ 5) $x_2 = \frac{num}{5}$

by scaling these features, the consumer ranges of values by dividing the maximum value



$0 \le x_1 \le 1, \quad 0 \le x_2 \le 1$

Get every features into approximately a $-1 \le x_i \le 1$ range

$0 \le x_1 \le 3$ ✓ $\quad -100 \le x_1 \le 100$ ✗

$-2 \le x_2 \le 0.5$ ✓ $\quad -0.0001 \le x_2 \le 0.0001$ ✗

Don't worry if the features are not exactly on the same scale in the same range of values.

Mean normalization 均值归一化:

replace x_i with $x_i - \mu_i$ to make features have approximtely zero mean

$x_1 = \frac{size - 1000}{2000}$ ⇒ average size = 1000

$x_2 = \frac{\#bedrooms - 2}{5}$ ⇒ average room number = 2

$-0.5 \le x_1 \le 0.5, \quad -0.5 \le x_2 \le 0.5$

$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$ ← avg value of x_1 in training set

s_1 ← range of $x_{1max} - x_{1min}$ or the standard of deviation

$x_2 \leftarrow \frac{x_2 - \mu_2}{s_2}$

(2)

Learning rate α

Debugging: make a plot with number of iterations on the x-axis.

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

Computing parameters analytically

Gradient descent: to minimize the cost function J(θ), with the iterative algorithm that take many steps, multiple iterations of gradient descent to converge the global minimum

Normal equation: a method to solve for θ analytically (一步求解)

m = the number of training samples
n = the number of features

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

$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \quad y = [\,]$ m - dimensional vector

e.g. if $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & x_1^{(m)} \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$

pinv(x'*x)*x'*y this code to minimize the cost function of theta for new linear regression
Feature scaling is not necessary of normal equation, but it is important for gradient descent.

Gradient descent	Normal equation
Need to choose α	No need to choose α
Need many iterations	No need to iterate
Work well even when n is large	Need to compute $(x^T x)^{-1}$
$O(kn^2)$	Slow if n is very large
N = 1,000,000	N = 100 / 1000

the cost of inverting matrix = $O(n^3)$

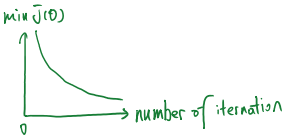
For most sophisticated algorithm, we always choose gradient descent.

What if $X^T X$ is non-invertible?

1. Redundant features
   X_1 = size in feet^2
   X_2 = sine in m^2
   In fact 1m = 3.28feet
2. Too many features n (m <= n)

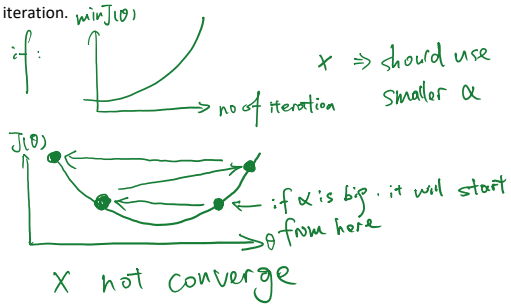
So we should use pinv() rather than inv(), the pinv() function will give you a value of thata if $X^T X$ is not invertible.

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

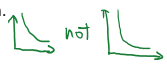


$J(\theta)$ should decrease after every iteration.

Except plot, automatic convergence test is also ok.
Declare convergence if J(θ) decreases by less than 10^-3 (or smaller) in one iteration.



For sufficiently smaller α, J(θ) should decrease on every interation.
But if α is too small, gradient descent can be slow to converge.



In summary:

1. If α is too small, slow convergence
2. If α is too large: J(θ) may not decrease on evert iteration; may not converge

how to test: 0.001 — 0.003 — 0.01 — 0.03 — 0.1 — 0.3 — 1 — ...

Features and polynomial regression

e.g. housing prices prediction

$h_\theta(x) = \theta_0 + \theta_1 \times \text{frontage} + \theta_2 \times \text{depth}$

$\Downarrow$

$h_\theta(x) = \theta_0 + \theta_1 \cdot \text{Area}$ (Area = frontage × depth)

We can choose different polynomial regression, quadratic function or cubic function, or else

$\theta_0 + \theta_1 x + \theta_2 x^2$ will come back down

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$ will not chd

how to fit (just like) Machinery of multivariate linear regression

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

$= \theta_0 + \theta_1 \text{size} + \theta_2 \text{size} + \theta_3 \text{size}$

$x_1 = \text{size}$, $x_2 = \text{size}^2$, $x_3 = \text{size}^3$

Feature scaling: size: $1 \sim 10^3$ (e.g.)
$\sim^2$: $1 \sim 10^6$
$\sim^3$: $1 \sim 10^9$

Other choices of features:

$h_\theta(x) = \theta_0 + \theta_1 \cdot \text{size} + \theta_2 \sqrt{\text{size}}$

第三周

2017年8月22日 22:09

Classification

Example
1. Email: spam / not span
2. Online transaction: fraudulent (yes / no)
3. Tumor: malignant / benign

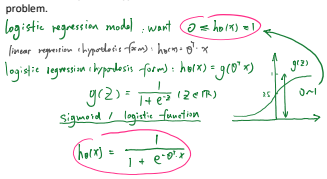
Y∈{0,1):
Threshold classifier output h_{theta} at 0.5:
If $h_{theta}(x)$ >= 0.5, predict y = 1
If < 0.5, predict y = 0

It is bad to use linear regression for classification problems.

Logistic regression: 0 <= $h_\theta(x)$ <= 1
This is a classification algorithm, not a regression algorithm, it is applied to settings where the label y is discrete value.

Hypothesis representation

A function to represent hypothesis when we have a classification problem.



Interpretation of Hypothesis Output
$h_{theta}(x)$ = estimated probability that y = 1 on input x

example if $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ tumorSize \end{bmatrix}$

when $h_\theta(x) = 0.7$

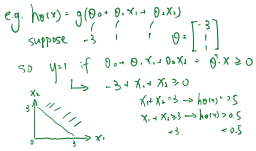
A patient with features x, the probability that y equals 1 is 0.7.
Tell patient that 70% change of tumor being malignant.

$h_\theta(x) = P(y=1 \mid x;\theta)$
Probability that y = 1, given x, parameterized by θ.
$P(y=0 \mid x,\theta) + P(y=1 \mid x,\theta) = 1$

Decision boundary

$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$
if z >= 0, then g(z) >= 0.5 → y = 1
z < 0, then g(z) < 0.5 → y = 0
so $\theta^T x \geq 0$, then y = 1
$\theta^T x < 0$, then y = 0

So the decision boundary is the line that separates the area where y = 0 and where y = 1. It is created by our hypothesis function.



If it is non-linear decision boundary:
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$

The decision boundary is a property, not a training set. Training set are used to fit the parameter data.

Here, all the theta are set to be known beforehand, so the next course will talk about how to automatically choose the parameters theta so that given a training set we can automatically fit the parameters to our data.

Logistical regression model

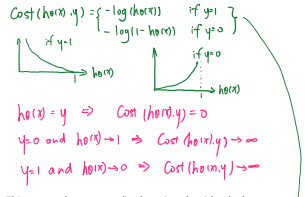
Cost function

Define the optimization objective or the cost function to fit the parameter.

training set $\{x^{(1)}, y^{(1)}, \dots, x^{(m)}, y^{(m)}\}$ m examples
$x \in \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix}$ n features $x_0 = 1$, $y \in \{0,1\}$
hypothesis $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$
Cost $(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2$

Non-convex function has a lot local optimum and not guarantee to the global minimum.

The penalty that the algorithm pays if it upwards the value of h(x)



$h_\theta(x) = y \Rightarrow Cost(h_\theta(x), y) = 0$
$y=0$ and $h_\theta(x) \to 1 \Rightarrow Cost(h_\theta(x), y) \to \infty$
$y=1$ and $h_\theta(x) \to 0 \Rightarrow Cost(h_\theta(x), y) \to \infty$

This means that we penalize learning algorithm by large cost.

Simplified cost function and gradient descent

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

Logistic regression cost function

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

The cost function can be derived from statistics using the principle of maximum likelihood estimation (最大似然估计), which is an idea in statistics for how to efficiently find parameters' data for different models, and also has nice property that it is convex.

To fit parameters θ, we need to find the minimum of J(θ) $\min_\theta J(\theta)$ → get θ
To make a prediction given new x
Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Gradient descent
$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))]$
want $\min_\theta J(\theta)$
repeat:
$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
$\theta_j = \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ vectorized implementation: $\theta := \theta - \alpha \frac{1}{m} X^T (g(X\theta) - \vec{y})$
(Simultaneously update for all j)

For linear regression, $h_\theta(x) = \theta^T x$
For logistic regression, $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$
a vectorized implementation for J(θ)
$J(\theta) = \frac{1}{m}(-y^T \log(h) - (1-y)^T \log(1-h))$

Advanced optimization algorithms and concepts

So as to minimize the cost function J of θ for logistic regression

cost function J(θ), want $\min_\theta J(\theta)$:
Given θ, we have code can compute
- J(theta)
- $\frac{\partial}{\partial \theta_j} J(\theta)$

Optimization algorithms:
- Gradient descent
- Conjugate gradient (共轭梯度法)
- BFGS ()
- L-BFGS

Advantages:
- No need to manually pick α
- Often faster than gradient descent

Disadvantages:
- More complex

Example: $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$
$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$
$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$
$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$

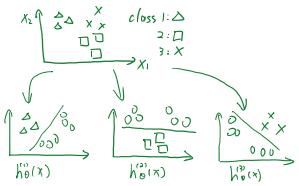
```
function [jVal, gradient] = contFunction(theta)
    jVal = (theta(1) - 5)2 + (theta(2) - 5)2;
    gradient = zeros(2,1);
    gradient(1) = 2*(theta(1) - 5);
    gradient(2) = 2*(theta(2) - 5);

options = optimset('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag] = fminunc(@contFunction,
initialTheta, options);
 % exitFlag > 0, mean converge to x; exitFlag = 0, mean overtake
iteration times; exitFlag < 0, mean not converge to x
```

Multiclass classification: one-vs-all

Example:
1. email foldering / tagging: work, friends, family, hobby
2. Medical diagrams: not ill, cold, flu
3. Weather: sunny, cloudy, rain, snow

y = 1, y = 2, y = 3...



$h_\theta^{(i)}(\lambda) = P(y=i \mid x;\theta)(i = 1,2,3)$
prediction = $\max_i(h_\theta^{(i)}(x))$

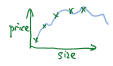
Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that y = i.
On a new input x, to make a prediction, pick the class i that maximize $\max_i(h_\theta^{(i)}(x))$

Solving the problem of overfitting

When we use linear regression or logistic regression, overfitting will be a problem.

As the size of the housing increases, the housing prices plateau, flattens out as we move to the right. So does not fit the training set very well, we call this problem: underfit or high bias

Overfitting: if we have too many, then then learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).



Two optional ways to address overfitting
1. Reduce number of features (select more important features)
   - Manually select which features to keep
   - Model selection algorithm
2. Regularization (正则化)
   - Keep all the features, but reduce magnitude / values of parameters θ_j
   - Work well when we have a lot of features, each of which contributes a bit to predicting y

Cost function

Regularization: small values for parameters $\theta_0, \dots, \theta_n$
- Simpler hypothesis
- Less prone to overfitting

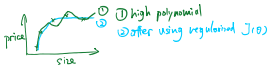
For example: housing
- Features: $x_1, \dots, x_{100}$
- Parameters: $\theta_0, \dots, \theta_{100}$

regularization term to shrink every single parameter

$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$
but not penalize θ_0

Our regularized cost function:
$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$ λ is regularization parameter

λ control a trade off between two goals:
1. Fit the training data
2. Keep parameters small



If λ is too large, algorithm reuslts in underfitting (fails to fit even the training set).
$\Rightarrow \theta_1 \approx 0, \theta_1 \approx 0 \cdots \theta_n \approx 0 \cdots \Rightarrow h_\theta(x) = \theta_0 \Rightarrow$ underfit (even a flat line)

Regularized linear regression

1. Gradient descent

Repeat:
$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$
$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m}\theta_j\right]$
$(j = 1, 2, \cdots n)$
$\Downarrow$
$\theta_j = \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ (especially $1 - \alpha\frac{\lambda}{m} < 1$)

It means, for example, if $1 - \alpha\frac{\lambda}{m} = 0.99$, it will shrink θ_j a little bit towards zero

2. Normal equation

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

$\min_\theta J(\theta)$
$\theta = (X^T X + \lambda \begin{bmatrix} 0 & & \\ & \ddots & \\ & & 1 \end{bmatrix})^{-1} X^T y$ ((n+1)×(n+1) matrix)

Non-invertibility problem
Suppose m <= n
$\theta = (X^T X)^{-1} X^T y$
non-invertible, singular, degenerate
if λ > 0
$\theta = (X^T X + \lambda \begin{bmatrix} 0 & & \\ & 1 & \\ & & \ddots \end{bmatrix})^{-1} X^T y$ ((n+1)×(n+1) matrix)
with regularization, it will be invertable

Regularized logistic regression

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

repeat {
$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$
$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m}\theta_j\right]$ $(j = 1, \cdots n)$

Advanced optimization

```
Function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(θ);
    Gradient(1) = [code to compute ∂/∂θ0 J(θ)]; ← θ0
    …                                           ← θ1
```

第四周

2017年8月28日 23:38

Feature number: n
Example number: m

Non-linear hypotheses

Neural network

Example: computer vision

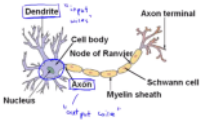
50 × 50 pixel images → 2500 pixels
n = 2500 (7500 if RGB)
$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2} \\ \vdots \\ \text{pixel 2500} \end{bmatrix}$
with grayscale image.
Quadratic features ($x_i \cdot x_j$): $\approx \frac{(n+1)n}{2} \approx 3$ million features

So linear regression is not a good way if the number of features n is very large.

Neurons and the Brain

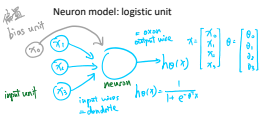
Origin: algorithms that try to mimic the brain

Neuron in the brain



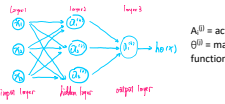
Model representation, part 1

Neuron model: logistic unit



Some terminology with the same meaning for the last videos
Sigmoid (logistic) activation function: $h_\theta(x)$
Theta: weight

Neural network



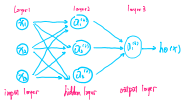
$A_i^{(j)}$ = activation of unit i in layer j
$\theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer j+1

$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$
$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$
$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$
$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$
If network has s_j units in layer j, and s_{j+1} units in layer j+1, then $\Theta^{(j)}$ will be of dimension $s_{j+1} * s_j + 1$

Model representation, part 2

Vectorization
$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \quad z_1^{(2)}$
$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \quad z_2^{(2)}$
$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \quad z_3^{(2)}$
$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$

$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \\ \cdots \end{bmatrix} \quad z^{(2)} = \Theta^{(1)}a^{(1)} \quad a^{(2)} = g(z^{(2)})$
$a_0^{(2)} = x_0^{(2)} = 1$, similarly $a_0^{(3)} = 1$
$z^{(3)} = \Theta^{(2)}a^{(2)} \quad h_\Theta(x) = a^{(3)} = g(z^{(3)})$



$A_i^{(j)}$ = activation of unit i in layer j
$\theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer j+1

Neural network learning its own features

Other network architecture
Architecture means how the different neurons are connected to each other.

Summary: $z^{(j)} = \Theta^{(j-1)} \cdot a^{(j-1)}$
$a^{(j)} = g(z^{(j)})$

Non-linear classification example: XOR / XNOR

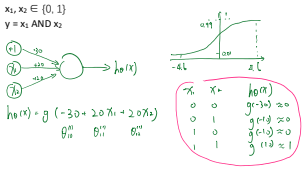
x_1, x_2 是二进制的 (0 或 1)。

$y = x_1$ XOR x_2 (当且仅当x1或x2有且仅有一个等于1时为真)

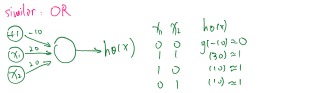
$y = x_1$ XNOR x_2 = not (x_1 XOR x_2)

Simple example: AND

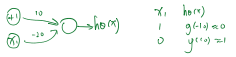
$x_1, x_2 \in \{0, 1\}$
$y = x_1$ AND x_2



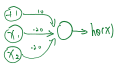
Similar: OR



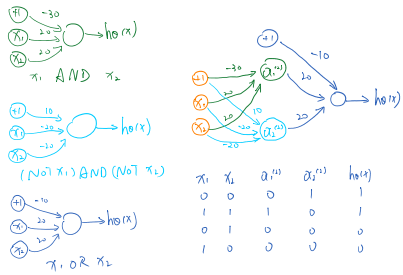
Simple example: negation (NOT x1)



Simple example: (NOT x1) AND (NOT x2)

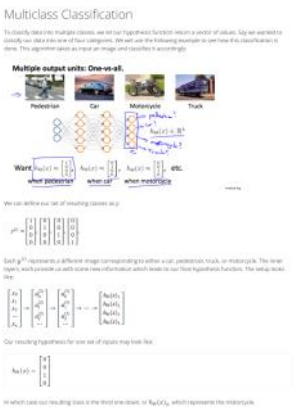


Putting it together: x1 XNOR x2



X1 XNOR X2 = NOT (X1 XOR X2) = (X1 AND X2) OR ((NOT X1) AND (NOT X2)) $(a \wedge b) \vee (\neg a \wedge \neg b)$
X1 XOR X2 = (NOT X1 AND X2) OR (X1 AND NOT X2) $(\neg a \wedge b) \vee (a \wedge \neg b)$

Multi-class classification



第五周

2017年9月4日 1:01

Cost function and backpropagation

L = total no. of layers in network
S_l = no. of units (not counting bias unit) in layer l
K = number of output units / classes

① Binary classification
$y = 0$ or 1 . 1 output unit $\Rightarrow \bigcirc \rightarrow h_\Theta(x)$

② Multi-class classification (K classes)
$y \in \mathbb{R}^K \Rightarrow \begin{bmatrix}1\\0\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\end{bmatrix}$ $K=3$
car bike plane
K output units . $h_\Theta(x) \in \mathbb{R}^K$ (generally $K>3$)

Cost function
Logistic regression:

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

m examples n features
do not regularize the bias term Θ_0.

Neural network:

$h_\Theta(x) \in \mathbb{R}^K$, $(h_\Theta(x))_i$ = i^{th} output

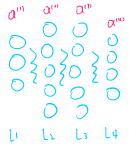
$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}(\Theta_{ji}^{(l)})^2$$

Backpropagation algorithm: a neural-network terminology for minimizing our cost function

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

Gradient computation
Given one training example (x, y)
Forward propagation:

$a^{(1)} = x$
$z^{(2)} = \Theta^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)})$ (+ $a_0^{(2)}$)
$z^{(3)} = \Theta^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(3)})$ (+ $a_0^{(3)}$)
$z^{(4)} = \Theta^{(3)} a^{(3)}$
$a^{(4)} = h_\Theta(x) = g(z^{(4)})$



Intuition: $\delta_j^{(l)}$ = 'error' of code j in layer l
For each output unit (layer L = 4)

$\delta_j^{(4)} = a_j^{(4)} - y_j$ $\quad$ $(h_\Theta(x))_j$

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

$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$ $\quad$ $a^{(2)} .* (1-a^{(2)})$

$g(z) = \frac{1}{1+e^{-z}} \Rightarrow g'(z) = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1+e^{-z}-1}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2} = y - y^2 = y(1-y)$

no $\delta^{(1)}$ because 1st layer corresponds to input layer . observed as training set so have no error

$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$ (when ignore $\lambda = 0$: regularization)

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}),\ldots, (x^{(m)}, y^{(m)})\}$
Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j) (use to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)
Upper case: Δ (accumulator) Lower case: δ

```
For i = 1 to m
    Set a(1) = x(1)
    Perform forward propagation to compute a(l) for l = 2, 3,…,L
    Using y(i), compute δ(L) = a(L) − y(i)
    Compute δ(L-1), δ(L-2),…, δ(2)
    Δij(l) := Δij(l) + aj(l) δi(l+1)
```

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

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

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

The capital-delta matrix D is used as an accumulator to add up our values as we go along and eventually compute our partial derivative.

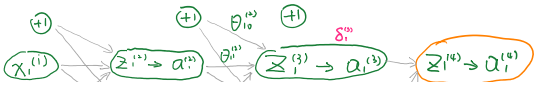
Suppose you have two training examples $(x^{(1)}, y^{(1)})$ and $(x^{(2)}, y^{(2)})$. Which of the following is a correct sequence of operations for computing the gradient? (Below, FP = forward propagation, BP = back propagation).

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

Backpropagation intuition

Forward propagation

z: the weighted sum of inputs of the input units then apply the sigmoid function $\Rightarrow$ a



Backpropagation in practice

Unrolling parameters from matrices into vectors

```
Function [jVal, gradientvec] = costFunction(thetavec)
optTheta = fminunc(@costFunction, initialTheta, options)
```

Neural Network (L=4)
θ(1),θ(2),θ(3) - matrices(Theta1, Theta2, Theta3)
D(1),D(2),D(3) - matrices(D1, D2, D3)
"unroll" into vectors

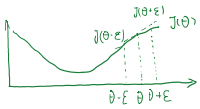
example
$S_1 = 10$ $S_2 = 10$ $S_3 = 1$
$\Theta^{(1)} \in \mathbb{R}^{11\times 11}$. $\Theta^{(2)} \in \mathbb{R}^{11\times 11}$. $\Theta^{(3)} \in \mathbb{R}^{1\times 11}$
$D^{(1)} \in \mathbb{R}^{10\times 11}$. $D^{(2)} \in \mathbb{R}^{10\times 11}$. $D^{(3)} \in \mathbb{R}^{1\times 11}$


```
thetaVec = [Theta1(:);Theta2(:);Theta3(:)];
Dvec = [D1(:);D2(:);D3(:)];
Theta1 = reshape(thetaVec(1:110),10,11);
Theta2 = reshape(thetaVec(111:220),10,11)
Theta3 = reshape(thetaVec(221:231),1,11);
```

```
Function [jVal, gradientvec] = costFunction(thetavec)
    From thetavec, get θ(1),θ(2),θ(3)
    Use forward prop/backprop to compute D1,D2,D3 and J(θ)
    Unroll D1,D2,D3 to get gradientvec
```

Gradient checking

Numerical estimation of gradients



$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$

Implement:

```
gradapprox = (J(theta + epsilon) - J(theta - epsilon)) / (2*epsilon)
```

$\theta \in \mathbb{R}^n$ θ is unrolled version of $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$

$\theta = [\theta_1, \theta_2, \theta_3 \cdots \theta_n]$

$$\frac{\partial}{\partial \theta_i} J(\theta) \approx \frac{J(\theta_1, \theta_2, \cdots, \theta_i+\varepsilon, \cdots, \theta_n) - J(\theta_1, \theta_2, \cdots, \theta_i-\varepsilon, \cdots, \theta_n)}{2\varepsilon}$$

```
Epsilon = 1e-4
For i = 1:n // n is the dimension of our parameter of vector theta
    Thetaplus = theta;
    Thetaplus(i) = thetaplus(i) + epsilon;
    Thetaminus = theta;
    Thetaminus(i) = thetaminus(i) - epsilon;
    Gradapprox(i) = (J(thetaplus) - J(thetaminus)) / (2*epsilon);
end
```

$\begin{bmatrix}\theta_1\\\theta_2\\\vdots\\\theta_i+\varepsilon\\\vdots\\\theta_n\end{bmatrix}$

check that gradApprox ≈ DVec

Implementation note:
1. Implement backprop to compute Dvec (unrolled D1, D2, D3)
2. Implement numerical gradient check to compute gradapprox
3. Make sure they give similar values (≈)
4. Turn of gradient checking, using backprop code for learning

Important:
Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of costfunction(...)) your code will be very slow.

Random initialization - Initial value of θ

For gradient descent and advanced optimization method, need initial value of θ

Optttheta = fminunc(@costfunction, initialtheta, options)

Consider gradient descent

Zero initialization

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

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

$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta)$

After each update, parameters corresponding to inputs going into each of two hidden units are identical, so one way to avoid it is to **break** it

Symmetry breaking - 对称性破缺 (one way for random initialization)
Initialize each $\Theta_{ij}^{(l)}$ to a random value in [-ε,ε]

```
Theta1 = rand(10,11) * (2*init_epsilon) - init_epsilon
Theta2 = rand(10,11) * (2*init_epsilon) - init_epsilon
Theta3 = rand(1,11) * (2*init_epsilon) - init_epsilon
```

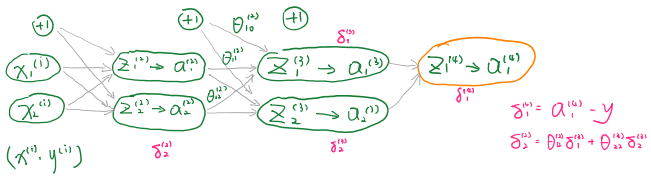
pick a random number r: rand(1,1) * (2 * init_epsilon) - init_epsilon
set $\Theta_{ij}^{(l)} = r$ for all i,j,l
because this fails to break symmetry

Put it together

Pick a network architecture (connectivity pattern between neurons)

Application of Neural Network - autonomous driving

Forward propagation



$(x^{(i)}, y^{(i)})$

$z_1^{(2)} = \Theta_{10}^{(1)} \cdot 1 + \Theta_{11}^{(1)} \cdot a_1^{(2)} + \Theta_{12}^{(1)} \cdot a_2^{(2)}$ (forward propagation)

When the algorithm is applying forward propagation, what is backpropagation doing?

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

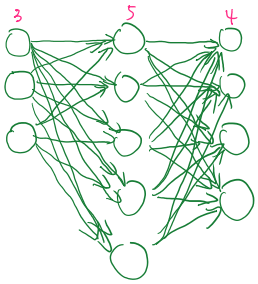
Focus on a single example $x^{(i)}$, $y^{(i)}$, the case of 1 output unit and ignoring regularization ($\lambda = 0$)

$\delta_j^{(l)}$ = error of cost for $a_j^{(l)}$ (unit j in layer l). formally $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(i)$ (for $j \geq 0$)

where $\text{cost}(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \log(1 - h_\Theta(x^{(i)}))$

Put it together

Pick a network architecture (connectivity pattern between neurons)



No. of input units: dimension of features $x^{(i)}$
No. of output units: number of classes
Reasonable default: 1 hidden layer, or if >1 hidden layer, have **same no.** of hidden units in every layer (usually the more the better)

Training a neural network

1. Randomly initialize weight
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

To calculate backprop, we use loop
For i = 1:m ← $(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})$
Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
(get activations $a^{(l)}$, and delata terms $\delta^{(l)}$ for l = 2,...,L)

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

compute $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical esimate of gradient of $J(\Theta)$
   Then, disable gradient checking code
6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

第六周

2017年9月17日 9:54

Evaluating a learning algorithm

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

If it has unacceptably large error, here is some solutions:

1. Get more training examples
2. Try smaller sets of features
3. Try getting additional features
4. Try adding polynomial features → 多项式
5. Try decreasing lambda
6. Try increasing lambda

Machine learning diagnostic:
Diagnostic: a test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

Diagnostic can take time to implement, but doing so can be a very good use of your time.

Dataset: size price } 100
前 70% : training set
后 30% : test set
training: $(x^{(m)}, y^{(m)})$
test: $(x^{(m_{test})}_{test}, y^{(m_{test})}_{test})$

Evaluating your hypothesis
- Learn parameter theta from training data (minimize training error J(θ))
- Compute test set error

$J_{test}(\theta)=\frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta(x^{(i)}_{test})-y^{(i)}_{test})^2$ linear regression

$J_{test}(\theta)=-\frac{1}{m_{test}}\sum_{i=1}^{m_{test}}[y^{(i)}_{test}\log(h_\theta(x^{(i)}_{test}))+(1-y^{(i)}_{test})\log(1-h_\theta(x^{(i)}_{test}))]$ logistic regression

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

$\text{test error}=\frac{1}{m_{test}}\sum_{i=1}^{m_{test}}error(h_\theta(x^{(i)}_{test}),y^{(i)})$

Model selection

Degree of polynomial = d

d=1 $h_\theta(x)=\theta_0+\theta_1x$

d=2 $h_\theta(x)=\theta_0+\theta_1x+\theta_2x^2$

⋮

d=10 $h_\theta(x)=\theta_0+\theta_1x+\theta_2x^2+\dots+\theta_{10}x^{10}$

Evaluating your hypothesis

} training set 60% $(x^{(m)}, y^{(m)})$
} cross validation 20% $(x^{(m)}_{cv}, y^{(m)}_{cv})$ ← use to choose
} test set 20% $(x^{(m)}_{test}, y^{(m)}_{test})$

Train / validation / test error

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

cross validation error $J_{cv}(\theta)=\frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x^{(i)}_{cv})-y^{(i)}_{cv})^2$

test error $J_{test}(\theta)=\frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta(x^{(i)}_{test})-y^{(i)}_{test})^2$

We can calculate three separate error values for the three different sets using the following method:

1. Optimize the parameters in θ using the training set for each polynomial degree
2. Find the polynomial degree d with the least error using the cross validation set
3. Estimate the generalization error using the test set with $J_{test}(\theta)^{(d)}$, d = theta from polynomial with lower error

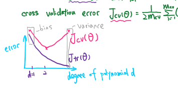
Bias VS Variance

Underfitting or overfitting problem

欠拟合 过拟合

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

cross validation error $J_{cv}(\theta)=\frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x^{(i)}_{cv})-y^{(i)}_{cv})^2$

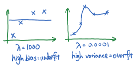


Bias (underfit): $J_{train}(\theta)$ will be high $J_{cv}(\theta)\approx J_{train}(\theta)$

Variance (overfit): $J_{train}(\theta)$ will be low $J_{cv}(\theta)\gg J_{train}(\theta)$

Regularization and Bias/Variance

underfit λ 很大
overfit λ 很小



Linear regression with regularization

model: $h_\theta(x)=\theta_0+\theta_1x+\theta_2x^2+\theta_3x^3+\theta_4x^4$

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

Choosing the regularization parameter λ

try $\lambda=0 \rightarrow \min_\theta J(\theta)\rightarrow\theta^{(1)}\rightarrow J_{cv}(\theta^{(1)})$

$0.01 \rightarrow \min_\theta J(\theta)\rightarrow\theta^{(2)}\rightarrow J_{cv}(\theta^{(2)})$

⋮

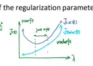
$10 \rightarrow \min J(\theta)\rightarrow\theta^{(m)}\rightarrow J_{cv}(\theta^{(m)})$

Bias/Variance as a function of the regularization parameter λ

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

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

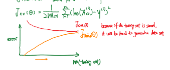
$J_{cv}(\theta)=\frac{1}{2m_{cv}}\sum_{i=1}^{m}(h_\theta(x^{(i)}_{cv})-y^{(i)})^2$



Learning curve

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

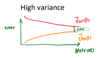
$J_{cv}(\theta)=\frac{1}{2m_{cv}}\sum_{i=1}^{m}(h_\theta(x^{(i)}_{cv})-y^{(i)})^2$



High bias - learning curve

If a learning algorithm is suffering from high bias, getting more training data will not help much because the high error will still high.

High variance



If a learning algorithm is suffering from high variance, getting more training data is likely to help.

variance – overfit
bias – underfit

If it has unacceptably large error, here is some solutions:

1. Get more training examples → fixed high variance
2. Try smaller sets of features → fixed high variance
3. Try getting additional features → fixed high bias
4. Try adding polynomial features → bias
5. Try decreasing lambda → bias
6. Try increasing lambda → variance

Neural networks and overfitting

Small neural network: fewer parameters are prone to underfitting
Large neural network: more parameters are prone to overfitting (use regularization to solve)

随堂测

Suppose you fit a neural network with one hidden layer to a training set. You find that the cross validation error is much larger than the training error. Is increasing the number of the hidden units likely to help?
Answer: No, because it is currently suffering from high variance, so adding hidden units is unlikely to help.

Build a spam classifier

Supervised learning
x = features of email
y = spam(1) or not spam(0)

From: patrick@163.com
To: ppap@163.com
Subject: Now?!
Deal of the week: Buy now ~

$x_j=\begin{cases}1 & \text{if word } j \text{ appears in email}\\ 0 & \text{otherwise}\end{cases}$

$X=\begin{bmatrix}0\\1\\1\\1\end{bmatrix}$ Andrew, buy, deal, now

In practice, take most frequently occurring n words (10000 to 50000) in training set, rather than manually pick 100 word.

How to improve the accuracy of this classifier?

1. Collect lots of data ("honeypot" project)
2. Develop sophisticated features (e.g. use email header data)
3. Develop algorithm to process input in different ways (e.g. recognize misspellings in spam)

Error analysis

1. Start with a simple algorithm that you can implement quickly
   Implement it and test it on your cross-validation data
2. Plot learning curves to decide if more data, more features
3. Error analysis: manually examine the examples (in cross-validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

example: M_{cv} = 500 examples in cross-validation set
algo misclassifies 100 emails
manually examine the 100 errors, and categorize them based on:
1) what type of email it is
2) what cues (features) you think would have helped the algo classify them correctly.

1) pharma 5 emails
Steal password 4 ~
other

2) deliberate misspellings 4 places
unusual punctuation 12 places

The importance of numerical evaluation

1. Should discount/discounts be treated as the same word?
2. Can use "stemming" software (e.g. "Porter stemmer"): universe / university
   Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.
   Need numerical evaluation (e.g. cross-validation error) of algorithm's performance with and without stemming 除词/词干

without stemming: 5% error with stemming: 3% error

e.g. distinguish upper vs. lower case (Mom/mom): 3.2% error

use error analysis on the cross-validation set!

Handling skewed data

Error metrics for skewed classes

误差度量 偏斜的类

Cancer classification example

Train logistic regression model $h_\theta(x)$ ($y = 1$ if cancer; $y = 0$ if not cancer)
Find that you got 1% error on test set (malignant vs benign)

if only 0.5% of patients have cancer

```
Function y = predictCancer(x)
    y = 0; %ignore x!
return
```

Precision / recall
y = 1 in presence of rare class that we want to detect

Actual class (columns), predicted class (rows):

	1	0
1	true positive	false positive
0	false negative	true negative

positive: 阳性
negative: 阴性

Precision 查准率
Of all patients where we predicted y = 1, what fraction actually has cancer?

$\frac{\text{True positive}}{\#\text{ predicted positive}}=\frac{\text{True pos}}{\text{True pos}+\text{false pos}}$

Recall 召回率
Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?

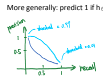
$\frac{\text{True pos}}{\#\text{ actual pos}}=\frac{\text{True pos}}{\text{True pos}+\text{false neg}}$

Precision and recall are better than just using classification error or classification accuracy.

Trading off precision and recall

logistic regression $0\leq h_\theta(x)\leq 1$
predict 1 if $h_\theta(x)\geq 0.7$
predict 0 if $h_\theta(x)<0.7$
suppose we want to predict y=1, only if very confident
→ higher precision, lower recall
suppose we want to avoid missing too many cases of cancer
→ higher recall, lower precision

More generally: predict 1 if $h_\theta(x)$ > threhold 阈值



F_1 Score (F Score)
How to compare precision/recall numbers?

	Precision(P)	Recall(R)	F1 Score
Algorithm1	0.5	0.4	0.444
Algorithm2	0.7	0.1	0.175
Algorithm3	0.02	1.0	0.0392

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

How to pick the value to use for the threshold:
Measure precision (P) and recall (R) on the cross validation set and choose the value of threshold which maximizes $2\frac{PR}{P+R}$

Data for machine learning

e.g. classify between confusable words

For breakfast I ate ___ eggs. {to, two, too}. {the

Algorithms:
- Perceptron (logistic regression) 感知器
- Winnow
- Memory-based
- Naïve Bayes

It's not who has the best algorithm that wins, the most data.

Large data rationale

Assume feature $x\in R^{n+1}$ has sufficient informa accurately

Example:
For breakfast, I ate _____ eggs.
Counterexample:
Predict housing price from only size (feet ^{2}) an

Useful test:
Given the input x, can a human expert confide

Use a learning algorithm with many paramete regression; neural network with many hidden

use a very large training set ⇒ not overfit

$\{x_1, x_2\}$

it's who has

tion to predict y

d no other features

ently predict y?

rs: logistic/linear
units.

第七周

2017年9月22日 2:35

Support vector machines (SVM)

Large margin classification

Optimization objective

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

$z = e^{-\theta^T x}$

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

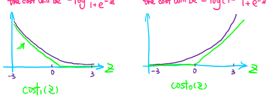
if $y=0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

cost: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

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

if $y=1$ (want $\theta^T x \gg 0$) the cost will be $-\log\frac{1}{1+e^{-z}}$

if $y=0$ (want $\theta^T x \ll 0$) the cost will be $-\log(1-\frac{1}{1+e^{-z}})$



Support vector machine

Logistic regression

$$\min_\theta J = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}\underbrace{(-\log h_\theta(x^{(i)}))}_{cost_1(\theta^T x^{(i)})} + (1-y^{(i)})\underbrace{(-\log(1-h_\theta(x^{(i)})))}_{cost_0(\theta^T x^{(i)})} + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

(first part: A; regularization part: B)

Support vector machine

$$\min_\theta J = \frac{1}{m}\sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)}) + \frac{\lambda}{2m}\sum_{i=0}^{n}\theta_j^2$$

Logistic regression: $A + \lambda B$

support vector machine: $CA + B$

C plays a similar role to $\frac{1}{\lambda}$

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

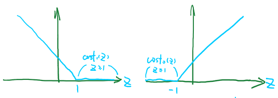
SVM hypothesis

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

Hypothesis

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

Large margin intuition 大间距分类器



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

If y=1, we want $\theta^T x$ >= 1 (not just >= 0) as supported margin factor

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

SVM decision boundary

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

whenever $y^{(i)}=1$: $\theta^T x^{(i)} \geq 1$

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

Let's set C is very large = 100, and the first term of cost function = 0

$\Rightarrow \min C \times 0 + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$

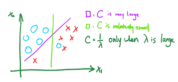
s.t. $\theta^T x^{(i)} \geq 1$, if $y^{(i)}=1$

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

Linearly separable case

Large margin classifier 大间距分类器

Large margin classifier in presence of outliers 异常点

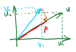


The mathematics behind large margin classification

Optimization problem => large margin classifier

Vector inner product 向量内积

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



norm of u: u的范数

$\|u\|$ = length of vector u (euclidean)

Pythagoras 毕达哥拉斯 theorem

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

p = length of projection v onto u

$u^T v = p \cdot \|u\| = u_1 v_1 + u_2 v_2$ ($p \in \mathbb{R}$)

$\begin{cases} p \cdot \|u\| > 0 & \text{if angle} < 90° \\ p \cdot \|u\| < 0 & \text{if angle} > 90° \end{cases}$

SVM decision boundary

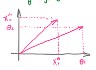
$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\|\theta\|^2$

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

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

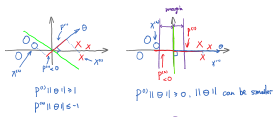
simplification: $\theta_0 = 0$, $n=2$

$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\sqrt{\theta_1^2+\theta_2^2} = \frac{1}{2}(\sqrt{\theta_1^2+\theta_2^2})^2 = \|\theta\|$



$\theta^T x^{(i)} = p^{(i)} \cdot \|\theta\|$

$= \theta_1 \cdot x_1^{(i)} + \theta_2 \cdot x_2^{(i)}$



A: $p^{(1)}\|\theta\| \geq 1$, $p^{(2)}\|\theta\| \leq -1$

B: $p^{(i)}\|\theta\| > 0$, $\|\theta\|$ can be smaller

Compared with A and B, we find the distance of margin $p^{(i)}$ should be large so as to make the $||\theta||$ smaller, because $\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$ should be small

Since we simplify $\theta_0=0$, which means that the decision boundary will pass the origin.

Kernels I

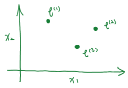
Non-linear decision boundary

Predict y = 1 if

$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \ldots >= 0$

$h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \cdots \\ 0 & \text{otherwise} \end{cases}$

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



Given x: $f_1 = similarity(x, l^{(1)}) = \exp(-\frac{\|x-l^{(1)}\|^2}{2\sigma^2})$

$f_2 = similarity(x, l^{(2)}) = \exp(-\frac{\|x-l^{(2)}\|^2}{2\sigma^2})$

$f_3 = similarity(x, l^{(3)}) = \exp(-\frac{\|x-l^{(3)}\|^2}{2\sigma^2})$

→ $k(x, l^{(i)})$

This function of similarity (相似度函数) called kernel function. Concretely, here, we call it Gaussian kernel.

Kernels and similarity

$f_1 = similarity(x, l^{(1)}) = k(x, l^{(1)}) = \exp(-\frac{\|x-l^{(1)}\|^2}{2\sigma^2}) = \exp(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2})$

if $x \approx l^{(1)}$: $f_1 \approx \exp(-\frac{0^2}{2\sigma^2}) \approx 1$

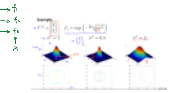
if x far from $l^{(1)}$: $f_1 = \exp(-\frac{(\text{large number})^2}{2\sigma^2}) \approx 0$

$l^{(1)} \to f_1$

$l^{(2)} \to f_2$

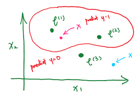
$l^{(3)} \to f_3$

↑ x



Predict '1' when

$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$



1. for this data x: $f_1 \approx 1$, $f_2 \approx 0$, $f_3 \approx 0$

and hypothesize $\theta_0 = -0.5$, $\theta_1 = 1$, $\theta_2 = 1$, $\theta_3 = 0$

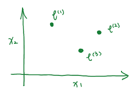
$\theta_0 + \theta_1 \cdot 1 + \theta_2 \cdot 0 + \theta_3 \cdot 0 = 0.5 > 0 \Rightarrow$ predict y=1

2. for this data x: $f_1, f_2, f_3 \approx 0$

$\to \theta_0 + \theta_1 \cdot 0 + \theta_2 \cdot 0 + \theta_3 \cdot 0 = -0.5 < 0 \Rightarrow$ predict y=0

Kernels II

Choosing the landmarks



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

?: where to get $l^{(1)}$ $l^{(2)}$ $l^{(3)}$

SVM with kernels

Given $(x^{(m)}, y^{(m)})$, choose $l^{(m)} = x^{(m)}$

$f_i = similarity(x, l^{(i)})$

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

for training example $(x^{(i)}, y^{(i)})$: $f_m^{(i)} = sim(x^{(i)}, l^{(m)})$

Hypothesis: given x, compute features $f \in \mathbb{R}^{m+1}$, $\theta \in \mathbb{R}^{m+1}$

Predict 'y=1' if $\theta^T f$ >= 0

Training:

$\min_\theta C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1-y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2}\sum_{j=1}^{m}\theta_j^2$ ← still do not regularize θ_0

$\sum_{j=1}\theta_j^2 = \theta^T \theta$ ($\|\theta\|^2$), $\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$ (ignore θ_0)

$\theta^T M \theta$ for efficiency

SVM parameters:

C (=1/λ)

Large C: lower bias, high variance

Small C: higher bias, low variance

σ^2

Large σ^2: features f_i vary more smoothly, higher bias, lower variance

Small σ^2: features f_i vary less smoothly, lower bias, higher variance

Using an SVM

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') $\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \geq 0$

Predict y = 1 if $\theta^T x$ >= 0 → n large, m small $x \in \mathbb{R}^{n+1}$

Gaussian kernel:

$f_i = \exp(-\frac{\|x-l^{(i)}\|^2}{2\sigma^2})$ where $l^{(i)} = x^{(i)}$, need to chose σ^2

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

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

(f_i; x1 = $x^{(i)}$, x2 = $x^{(j)}$)

```
Note: do perform feature scaling before using the Gaussian kernel
```

$\|x - l\|^2$ $v = x - l$

$\|v\|^2 = (x_1 - l_1)^2 + \cdots + (x_n - l_n)^2$

SVM have many optimization algo for θ

so must make sure: Mercer's Theorem 默塞尔定理

to make sure SVM packages can run correctly

Many off-the-shelf kernels available:

- Polynomial kernel $k(x,l) = (x^T l)^3$ or $(x^T l + 1)^3$ or $(x^T l + 5)^4 \Rightarrow (x^T l + \text{constant})^{degree}$
- String, chi-square 卡方, histogram intersection kernel...

Multi-class classification

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

SVM packages 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, ..., K), get $\theta^{(1)}$ ($y=1$), $\theta^{(2)}$ ($y=2$) ... $\theta^{(K)}$ ($y=K$)

And pick class i with largest $(\theta^{(i)})^T x$

Logistic regression vs SVM

n = number of features $x \in \mathbb{R}^{n+1}$

m = number of training examples

① if n large ($n \geq m$) n = 10.000, m = 10

use logistic regression, or SVM without kernal ('linear kernal')

② n small, m intermediate n = 1000, m = 10.000

use SVM with Gaussin kernal

③ n small m large n = 1000 m = 100.000

create more features, then use logistic or SVM without kernal

第八周

2017年10月2日　0:06

Clustering

Unsupervised learning

Application of clustering

1. Marketing segmentation
2. Social network analysis
3. Organize computing clusters
4. Astronomical data analysis

K-means algorithm

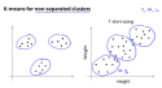
Input:

- K (number of clusters)
- Training set

Randomly initialize K clusters centroids

Repeat{
  For i = 1 to m
    c(i) := index (from 1 to K) of cluster centroid closest to x(i)
  For k = 1 to K
    μk = average(mean) of the ponts assigned to cluster k
}

K-means for non-separated clusters



Optimization objective

$c^{(i)}$ = index of cluster (1,...,K) to which example $x^{(i)}$ is currently assigned

μ_k = cluster centroid k

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

Optimization objective

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

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

Random initialization

1. should have $K < m$
2. randomly pick K training examples
3. Set $\mu_1,\dots,\mu_K$ equal to these K examples

Local optima

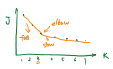
try many times for initialization

```
For i = 1 to 100 {
    randomly initialize K-means
    Run K-means. Get c(1),...,c(m), μ1,...,μK (centroids)
    Compute cost function (distortion)
    J(c(1),...,c(m),μ1,...,μK)
}
pick clustering that give lowest cost J(c,μ)
```

Choosing the numbers of clusters

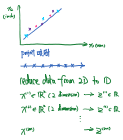
What is the right value of K?

Elbow method

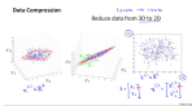


Dimensionality reduction - data compression

Data compression



Reduce data from 3D to 2D

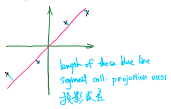


Dimensionality reduction - data visualization

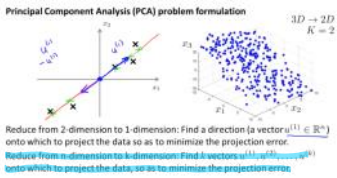
eg reduce data from 50 dimensions to 2 d

Principal component analysis (PCA)

PCA problem formulation

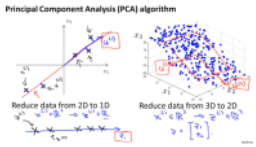


PCA is to find a surface onto which to project the data.



PCA algorithm

Data preprocessing



Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix" 协方差

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

Compute "eigenvectors" of matrix Σ:

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

Summary

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

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

Sigma = 1/m * X' * X

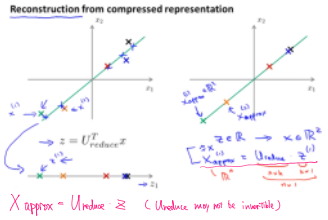
[U,S,V] = svd(Sigma);

Ureduce = U(:,1:k);

z = Ureduce' * x;

Applying PCA

Reconstruction from compressed representation



$X_{approx} = U_{reduce} z$ (Ureduce may not be invertible)

Choosing the number of principal component analysis

Average squared projection error 平均平方映射误差 $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}-x_{approx}^{(i)}\|^2$

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

Typically, choose k to be smallest value so that:

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}-x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2}\le 0.01 \text{ or } 0.05$$

99% of variance is retained 99%的变异性被保留

Choosing k

Algorithm:

Try PCA with k=1 ... 2 ... 3 ...

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

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

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

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

for given k

$1-\frac{\sum_{i=1}^{k}S_{ii}}{\sum_{i=1}^{n}S_{ii}}\le 0.01$ $\frac{\sum_{i=1}^{k}S_{ii}}{\sum_{i=1}^{n}S_{ii}}\ge 0.99$

Summary

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

Pick the smallest value of k for which

$\frac{\sum_{i=1}^{k}S_{ii}}{\sum_{i=1}^{n}S_{ii}}\ge 0.99$ (99% of variance retained)

Advice for applying PCA

Supervised learning speedup

$(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\dots,(x^{(m)},y^{(m)})$ $x^{(i)}\in\mathbb{R}^{10000}$

Extract inputs:

Unlabeled dataset: $x^{(1)}\dots x^{(m)}\in\mathbb{R}^{10000}$ → PCA → $z^{(1)}\dots z^{(m)}\in\mathbb{R}^{1000}$

New training set:

$(z^{(1)},y^{(1)}),\dots,(z^{(m)},y^{(m)})$ $h_\theta(x)=\frac{1}{1+e^{-\theta^T x}}$

Note: mapping $x^{(i)}$ -> $y^{(i)}$ should be defined by running PCA only on the training set. This mapping can be applied as well to the examples $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the cross validation and test sets.

Compression

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

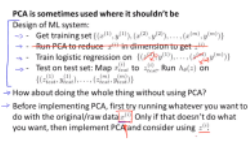
Visualization

Bad use of PCA: to prevent overfitting

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

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

Bad use of PCA: use it where it shouldn't be



第九周

2017年10月16日 13:17

Anomaly detection 异常检测

Density estimation

Problem estimation

Aircraft engine features — Dataset $\{x^{(1)}, \dots, x^{(m)}\}$

x_1 = heat generated; new engine: x_{test} ← anomalous?

x_2 = vibration intensity

to test x_{test} anomalous or not: model $p(x)$ → $p(x_{test}) < \varepsilon$ → flag anomaly; $p(x_{test}) \geq \varepsilon$ → OK

Anomaly detection examples:

Fraud detection:

- $x^{(i)}$ = features of users' activities
- Model p(x) from data
- Identify unusual users by checking which have $p(x) < \varepsilon$

Manufacturing

Monitoring computers in a data center:

- $x^{(i)}$ = features of machine i
- x_1 = memory use, x_2 = number of disk accesses
- x_3 = CPU load, x_4 = CPU load / network traffic

Gaussian distribution = normal distribution

Say $x \in \mathbb{R}$, if x is a distributed Gaussian with mean μ, variance σ^2

$x \sim N(\mu, \sigma^2)$ — σ (sigma) called standard deviation

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

the area = 1

Parameter estimation 参数估计

Dataset $\{x^{(1)}, \dots, x^{(m)}\}$, $x^{(i)} \in \mathbb{R}$, $x^{(i)} \sim N(\mu, \sigma^2)$

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

Apply Gaussian distribution to develop anomaly detection algorithm

Density estimation 密度估计

Training set $\{x^{(1)}, \dots, x^{(m)}\}$, each example is $x \in \mathbb{R}^n$, n features

$p(x) = p(x_1; \mu_1, \sigma_1^2) \cdots p(x_n; \mu_n, \sigma_n^2) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$, $x_1 \sim N(\mu_1, \sigma_1^2)$, $x_2 \sim N(\mu_2, \sigma_2^2)$

Algorithm:

1. Choose features x_i that you think might be indicative of anomalous examples
2. Fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$
   $\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$
   $\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$
3. Given new examples x, compute p(x):
   $p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j-\mu_j)^2}{2\sigma_j^2}\right)$
   anomaly if $p(x) < \varepsilon$ (epsilon)

Building an anomaly detection system

Developing and evaluation an anomaly detection system

The importance of real-number evaluation

When developing a learning algorithm (choosing features), 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 $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ (not anomalous)

Cross validation $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

Test set $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

Aircraft engines motivating example

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

Algorithm evaluation

Fit model p(x) on training set $x^{(1)}, \dots, x^{(m)}$

On a cross validation/test example x, predict

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

Possible evaluation metrics:

- True positive, false positive, false negative, true negative
- Precision准确率/recall召回率
- F1-score

Can also use cross validation set to choose parameter ε

Anomaly detection vs supervised learning

Anomaly detection	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	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
Many different "type" of anomalies. Hard for any algorithm to learn from positive examples what anomalies look like	
Future anomalies may look nothing like any of the anomalous examples we've seen so far	
Application: - Fraud detection - Manufacturing - Monitor machines in data center	Application: - Email spam classification - Weather prediction (sunny/rainy/etc.) - Cancer classification (benign, malignant)

Choose what features to use

Gaussian curve - bell shaped curve - normal distribution

Non-Gaussian features

$\log(x)$; way: $x \to \log(x + c)$, $x \to x^{\frac{1}{2}}$ or $x^{\frac{1}{3}}$

Error analysis for anomaly detection

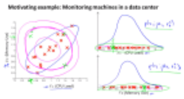
Want p(x) large for normal examples

p(x) small for anomalous examples

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

Question: suppose anomaly detection algorithm is performing poorly and outputs a large value of p(x) for many normal examples and for anomalous examples in your cross validation dataset. The way to help change the algorithm is to try coming up with more features to distinguish between the normal and the anomalous examples.

Multivariate Gaussian distribution 多元高斯分布



$x \in \mathbb{R}^n$, model p(x) all in one go

Parameter: $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (convariance matrix)

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

$|\Sigma|$ mean the determinant (行列式) of Σ | det(Sigma) — measure the variance or the variability of the features x_1, x_2

Multivariate Gaussian (Normal) examples

Anomaly detection using the Multivariate Gaussian Distribution

parameters μ, Σ

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

Parameter fitting: given tr set $\{x^{(1)}, \dots, x^{(m)}\}$

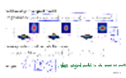
$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$, $\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$

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
   $p(x) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$
   flag an anomaly if $p(x) < \varepsilon$ (epsilon)

Relationship to original model

The contours of the Gaussian are always axis aligned



then original model is the same as multivariate gaussian

Original model	Multivariate gaussian
1. $p(x_1; \mu_1, \sigma_1^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$	1. $p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$
2. manually create features to capture anomalies where x_1, x_2 take unusual combinations of values → $x_3 = \frac{x_1}{x_2}$	2. automatically capture correlations between features
3. computationally cheaper (alternatively scales better to large n)	3. computationally more expensive
4. OK even if m is small	4. must have m > n, or else Σ is non-invertible ↓ exclude redundant features $x_1 = x_2$, $x_3 = x_4 + x_5$

Predicting Movie ratings

Problem formulation

User rates movies using zero to five stars

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)
Love	5	5	0	0
Romance	5	?	?	0
Cute	?	4	0	?
Nonstop	0	0	5	4
Swords	0	0	5	?

1. n_u = no. users
2. n_m = no. movies
- $r(i,j) = 1$ if user j has rated movie i, otherwise = 0
- $y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i,j) = 1$)

$n_u = 4$, $n_m = 5$

Content-based recommendations

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	x_1 (romance)	x_2 (action)
Love	5	5	0	0	0.9	0
Romance	5	?	?	0	1.0	0.01
Cute	?	4	0	?	0.99	0
Nonstop	0	0	5	4	0.1	1.0
Swords	0	0	5	?	0	0.9

commonly $x_0 = 1$; $x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$

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

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

$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix}$, $\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$, $(\theta^{(1)})^T x^{(3)} = 4.95$

Problem formulation

- $r(i,j) = 1$ if user j has rated movie i (0 otherwise)
- $y^{(i,j)}$ = rating by user j on movie i
- $\theta^{(j)}$ = parameter vector for user j
- $x^{(i)}$ = features vector from movie i
- For user j, movie i, predict rating: $(\theta^{(j)})^T (x^{(i)})$
- $m^{(j)}$ = no. of movies rated by user j

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

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

Optimization objective

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

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

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

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

Gradient descent update

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

$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i: r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} + \lambda \theta_k^{(j)}\right)$ (for k ≠ 0) — $\frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \dots, \theta^{(n_u)})$

Collaborative filtering

Optimization algorithm

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

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

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

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

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

Collaborating filtering

Given $x^{(1)}, \dots, x^{(n_m)}$ (and movie ratings), can estimate $\theta^{(1)}, \dots, \theta^{(n_u)}$ — 一个先有鸡还是先有蛋的故事 (chicken-egg story)

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

By rating a few movies myself, I will be helping the system learn better features and these features can be learnt by the system to make better movie prediction for everyone else (collaborating filtering)

Collaborative filtering algorithm

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

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

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

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

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

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

$\min_{x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$

exclude $x_0 = 1$, so $x \in \mathbb{R}^n$, $\theta \in \mathbb{R}^n$

Collaborative filtering algorithm

1. initialize $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values (symmetry breaking)
2. minimize $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$ using gradient descent
   $x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j: r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) \theta_k^{(j)} + \lambda x_k^{(i)}\right)$
   $\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i: r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} + \lambda \theta_k^{(j)}\right)$
3. for a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$

Low rank matrix factorization

Vectorization: low rank matrix factorization

predicted rating:

$(\theta^{(1)})^T (x^{(1)})$ $(\theta^{(2)})^T (x^{(1)})$ ⋯ $(\theta^{(n_u)})^T (x^{(1)})$

⋮

$(\theta^{(1)})^T (x^{(n_m)})$ $(\theta^{(2)})^T (x^{(n_m)})$ ⋯ $(\theta^{(n_u)})^T (x^{(n_m)})$

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

low rank matrix factorization 低秩矩阵分解 $X\Theta^T$

size(X) = $n_m \times n$, size(Θ) = $n_u \times n$

Finding related movies

For each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$

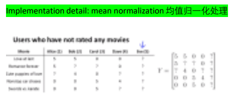
x_1 = romance, x_2 = action, x_3 = comedy, x_4 = …

How to find movies j related to movie i?

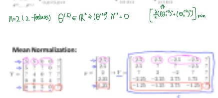
small $\|x^{(i)} - x^{(j)}\|$ → movie i and j are similar

e.g. 5 most similar movie to movie i: find the 5 movies j with the smallest $\|x^{(i)} - x^{(j)}\|$

Implementation detail: mean normalization 均值归一化处理



n=2 (2 features) $\theta^{(5)} \in \mathbb{R}^2 \Rightarrow (\theta^{(5)})^T x^{(i)} = 0$ — $\left[\frac{\lambda}{2}\left((\theta_1^{(5)})^2 + (\theta_2^{(5)})^2\right)\right]_{min}$



for user j, on movie i predict:

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

User 5 (Eve):

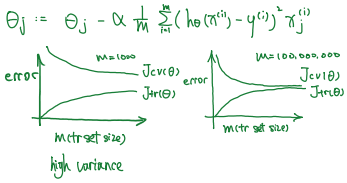
$\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $(\theta^{(5)})^T x^{(i)} + \mu_i$

第十周

2017年10月18日　18:01

Gradient descent with large datasets

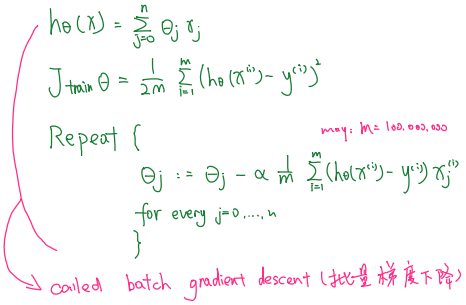
Learning with large datasets



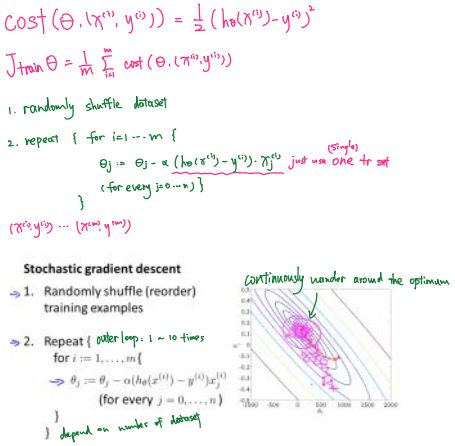
high variance

Stochastic gradient descent 随机梯度下降

Linear regression with gradient descent

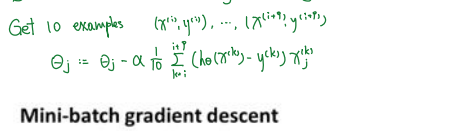


Stochastic gradient descent



Mini-Batch gradient descent - 小批量梯度下降

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



Mini-batch gradient descent

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

problem: must have good vectorized implementation

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

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

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

}

}

Stochastic gradient descent convergence

Checking for convergence

Batch gradient descent:

Advanced topics

Online learning - 在线学习机制

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.

price　logistic regression

Repeat forever { Get (x,y) corresponding to user

Update θ using (x,y). use once (x^(i), y^(i)), then push it

$\theta_j := \theta_j - \alpha (h_\theta(x) - y) x_j$ (j = 0, ..., n)

} Can adapt to change user perference

Other online learning example:

Product search (learning to search)

User searches for "iPhone 6"

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 $p(y = 1|x; \theta)$　predict CTR (click-through rate) 预估点击率

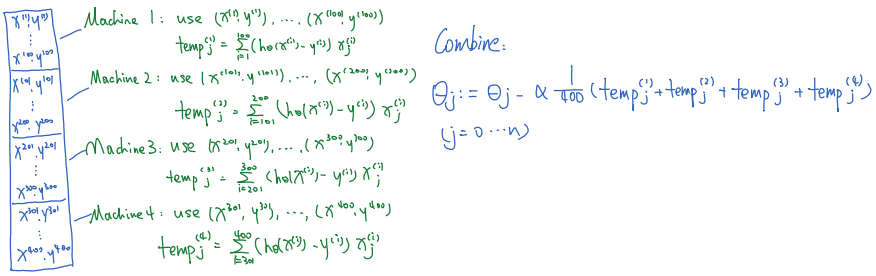
Use to show user the 10 phones they are most likely to click on

Other examples:

- choosing special offers to show user;
- customized selection of news articles;
- product recommendation

Map reduce and data parallelism 映射约减

Batch gradient descent $\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$



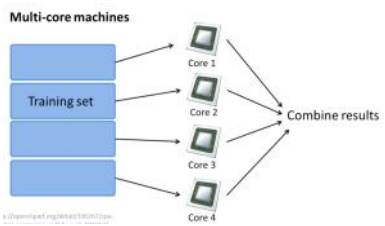
Map reduce and summation over the training set

Many learning algorithms can be expressed as computing sums of functions over the training set.

e.g. for advanced optimization, with logistic regression, need:

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

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



Plot $J_{train}(\theta)$ as function of the number of iterations of gradient descent

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

Stochastic gradient descent:

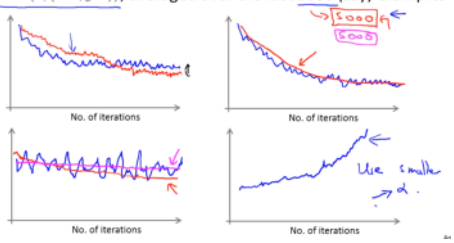
$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

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

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

Checking for convergence

Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples

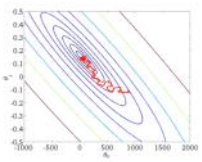


Stochastic gradient descent

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

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

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 $j = 0, \ldots, n$)
   }
   }



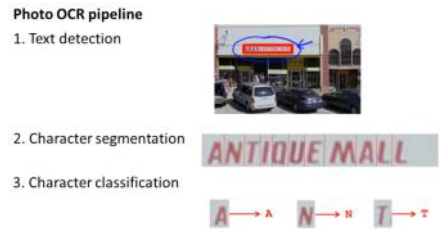
Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (E.g. $\alpha = \frac{const1}{iterationNumber + const2}$) $\alpha \to 0$

第十一周

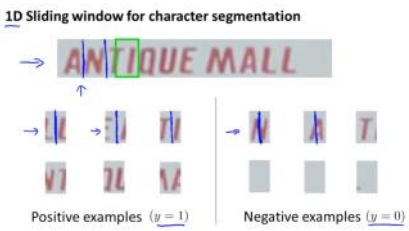
2017年10月19日　　1:51

Photo OCR - Photo optical character recognition 照片光学字符识别

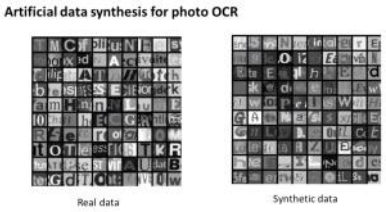
Problem description and pipeline 机器学习流水线

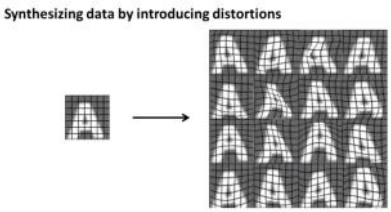


Sliding windows classifier



Getting lots of data and artificial data



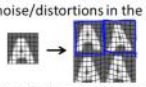


Summary: main topic

1. Supervised learning
   - Linear regression, logistic regression, neural network, SVM
2. Unsupervised learning
   - K-means, PCA, Anomaly detection
3. Special applications / special topics
   - Recommender systems, large scale machine learning
4. Advice on building a machine learning system
   - Bias / variance, regularization; deciding what to work on next: evaluation of learning algorithms, learning curves, error analysis, ceiling analysis

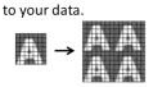
Synthesizing data by introducing distortions

- Distortion introduced should be representation of the type of noise/distortions in the test set.



- Audio: Background noise, bad cellphone connection

- Usually does not help to add purely random/meaningless noise to your data.



x_i = intensity (brightness) of pixel i
$x_i \leftarrow x_i +$ random noise

Discussion on getting more data

1. Firstly, 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
- "Crows source" (e.g. Amazon mechanical turk)

Ceiling analysis: what part of the pipeline to work on next - 上限分析

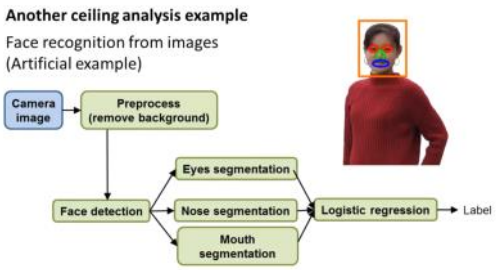
Image → text detection → character segmentation → character recognition

Q: which part is the most important?

Component	Accuracy
overall system	72%
text detection	89
character seg	90
character rec	100

Another ceiling analysis example 上限分析

Face recognition from images (artificial example)



Component	Accuracy
Overall system	85%
Preprocess (remove background)	85.1
Face detection	91
Eyes segmentation	95
Nose segmentation	96
Mouth segmentation	97
Logistic regression	100

0.1
5.9
4
1
1
3