Table of Contents

序

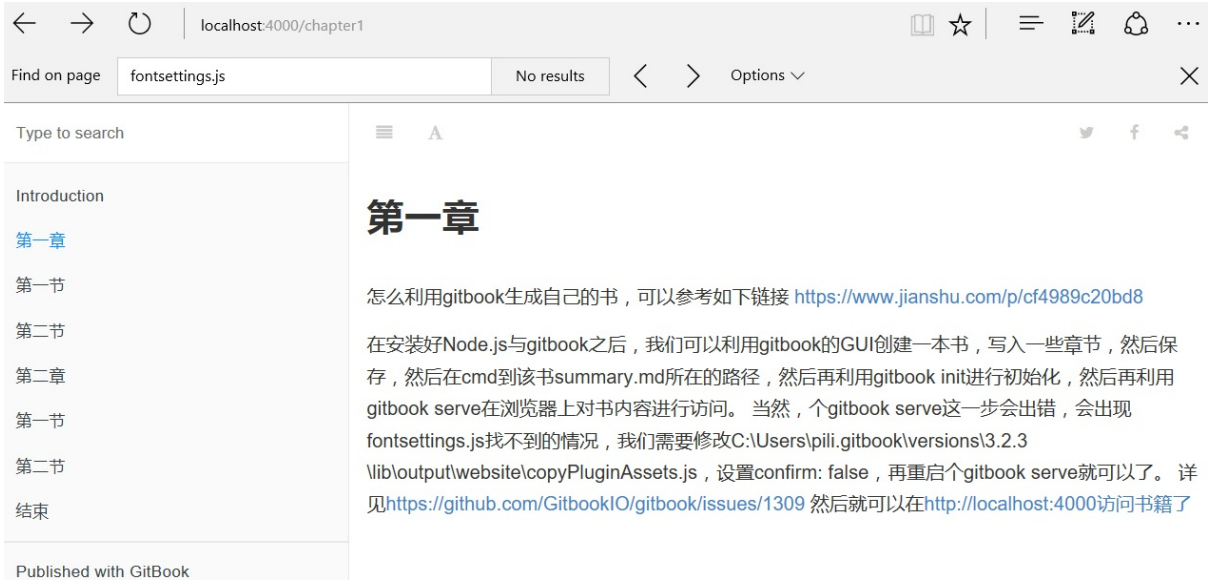
日知录是为效法顾炎武的《日知录》而做，里面会记录自己在学习与工作中学到的东西，该书的目的是为了打造一本属于自己的百科全书，也是自己思想体系的体现。最终里面的每一章可能代表一个方向与学科，比如C++，C#最终会成为一章，经济，中国史会成为一章。如果经济学里面记录的内容不多，就编成一章，如果内容多了，则会编成多章，比如宏观经济学，微观经济学，计量经济学。这样，最终一门大的学科，比如，经济学，就编成了一卷。

第一章:工具

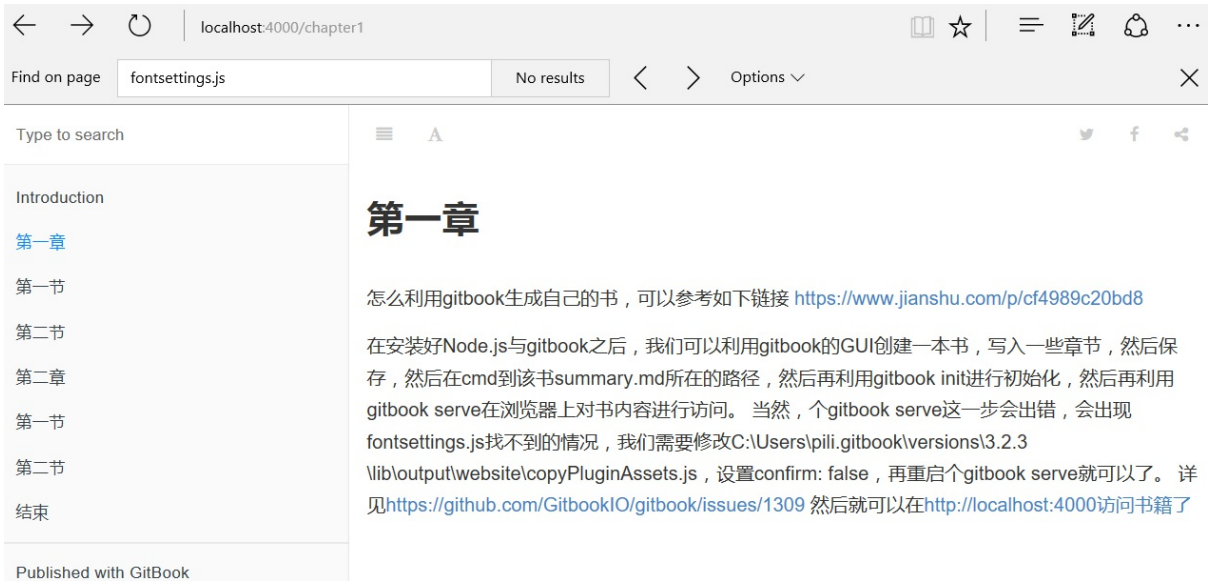
第一节 Gitbook的安装与使用

Gitbook是Github旗下的产品，它提供书籍的编写与管理的功能，一个核心特征就是，它管理书也像管理代码一样，可以对数据进行fork,建立新的branch,使得书也可以进行版本迭代。也可以与多人合作，来编辑，更新书籍，适合单人创作或者多人共同创作。怎么利用gitbook生成自己的书，可以参考如下链接 https://www.jianshu.com/p/cf4989c20bd8
https://www.cnblogs.com/Lam7/p/6109872.html

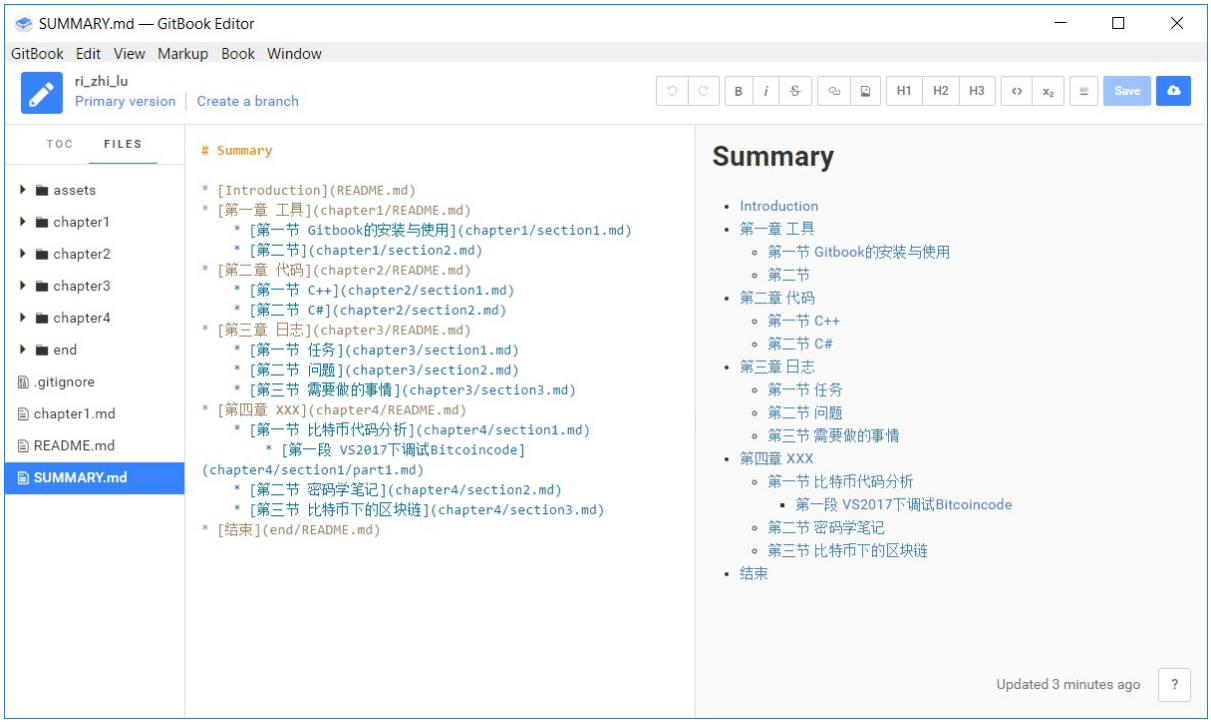
在安装好Node.js与gitbook之后，我们可以利用gitbook的GUI创建一本书，写入一些章节，然后保存，然后在cmd到该书summary.md所在的路径，然后再利用gitbook init进行初始化，然后再利用gitbook serve在浏览器上对书内容进行访问。当然，个gitbook serve这一步会出错，会出现fontsettings.js找不到的情况，我们需要修改C:\Users\pili.gitbook\versions\3.2.3\lib\output\website\copyPluginAssets.js，设置confirm: false，再重启个gitbook serve就可以了。详见https://github.com/GitbookIO/gitbook/issues/1309 然后就可以在 http://localhost:4000 访问书籍了



在安装好Node.js与gitbook之后，我们可以利用gitbook的GUI创建一本书，写入一些章节，然后保存，然后在cmd到该书summary.md所在的路径，然后再利用gitbook init进行初始化，然后再利用gitbook serve在浏览器上对书内容进行访问。当然，个gitbook serve这一步会出错，会出现fontsettings.js找不到的情况，我们需要修改C:\Users\pili.gitbook\versions\3.2.3\lib\output\website\copyPluginAssets.js，设置confirm: false，再重启个gitbook serve就可以了。详见https://github.com/GitbookIO/gitbook/issues/1309 然后就可以在 http://localhost:4000 访问书籍了



如果想让章节有层次感的显示，则可以在summary.md中设置



Error: Error with command "svgexport"

在cmd中安装：npm install svgexport -g

生成pdf

gitbook pdf ./ ./ML2.pdf

换行

一行之后加两个以上的空格。

第二节 编辑数学公式

mathjax

可以参考https://598753468.gitbooks.io/tex/content/fei_xian_xing_fu_hao.html
数学公式的编辑类似于Latex.

需要安装mathjax

```
npm install mathjax
安装特定版本的npm install mathjax@2.6.1
```

首先在书籍project的最顶成新建一个book.json,内容如下

```
{
    "gitbook": "3.2.3",
    "plugins": ["mathjax"],
    "links": {
        "sidebar": {
            "Contact us / Support": "https://www.gitbook.com/contact"
        }
    },
    "pluginsConfig": {
    "mathjax":{
            "forceSVG": true
        }
        }
}
```

然后再用gitbook install命令安装mathjax

安装之后gitbook就出现编译错误了，也不能编译生成pdf文件。不论mathjax是哪个版本，从2.5开始都出错。下面选择用katex

gitbook pdf ./ mybook.pdf（./ mybook.pdf 之间有空格）

http://ldehai.com/blog/2016/11/30/write-with-gitbook/

no such file fontsettings.js,在上一节有讲怎么处理。

安装Katex

https://github.com/GitbookIO/plugin-katex

1. 在你的书籍文件夹里创建book.json
2. 里面写入如下内容

```
{
    "plugins": ["katex"]
}
```

4. 运行安装 gitbook install就安装好了(安装很慢，一个小时),可以换个地方安装，只要把book.json换个地方，再在这个folder安装gitbook install，然后把node_modules 拷到你书籍所在的folder就可以。Katex支持的公式https://khan.github.io/KaTeX/docs/supported.html https://utensil-site.github.io/available-in-katex/

中文在cmd中乱码的问题：

1. 打开cmd，输入chcp 65001chcp 65001
2. 右击cmd上方，选择属性-->字体-->SimSun-ExtB就可以显示了。

语法高亮

```
Supported languages
This is the list of all 120 languages currently supported by Prism, with their corresponding
alias, to use in place of xxxx in the language-xxxx class:



Markup - markup
CSS - css
C-like - clike
JavaScript - javascript
ABAP - abap
ActionScript - actionscript
Ada - ada
Apache Configuration - apacheconf
APL - apl
AppleScript - applescript
AsciiDoc - asciidoc
ASP.NET (C#) - aspnet
AutoIt - autoit
AutoHotkey - autohotkey
Bash - bash
BASIC - basic
Batch - batch
Bison - bison
```

```
Brainfuck - brainfuck
Bro - bro
C - c
C# - csharp
C++ - cpp
CoffeeScript - coffeescript
Crystal - crystal
CSS Extras - css-extras
D - d
Dart - dart
Diff - diff
Docker - docker
Eiffel - eiffel
Elixir - elixir
Erlang - erlang
F# - fsharp
Fortran - fortran
Gherkin - gherkin
Git - git
GLSL - glsl
Go - go
GraphQL - graphql
Groovy - groovy
Haml - haml
Handlebars - handlebars
Haskell - haskell
Haxe - haxe
HTTP - http
Icon - icon
Inform 7 - inform7
Ini - ini
J - j
Jade - jade
Java - java
Jolie - jolie
JSON - json
Julia - julia
Keyman - keyman
Kotlin - kotlin
LaTeX - latex
Less - less
LiveScript - livescript
LOLCODE - lolcode
Lua - lua
Makefile - makefile
Markdown - markdown
MATLAB - matlab
MEL - mel
Mizar - mizar
Monkey - monkey
NASM - nasm
nginx - nginx
```

```
Nim - nim
Nix - nix
NSIS - nsis
Objective-C - objectivec
OCaml - ocaml
Oz - oz
PARI/GP - parigp
Parser - parser
Pascal - pascal
Perl - perl
PHP - php
PHP Extras - php-extras
PowerShell - powershell
Processing - processing
Prolog - prolog
.properties - properties
Protocol Buffers - protobuf
Puppet - puppet
Pure - pure
Python - python
Q - q
Qore - qore
R - r
React JSX - jsx
Reason - reason
reST (reStructuredText) - rest
Rip - rip
Roboconf - roboconf
Ruby - ruby
Rust - rust
SAS - sas
Sass (Sass) - sass
Sass (Scss) - scss
Scala - scala
Scheme - scheme
Smalltalk - smalltalk
Smarty - smarty
SQL - sql
Stylus - stylus
Swift - swift
Tcl - tcl
Textile - textile
Twig - twig
TypeScript - typescript
Verilog - verilog
VHDL - vhdl
vim - vim
Wiki markup - wiki
Xojo (REALbasic) - xojo
YAML - yaml
```

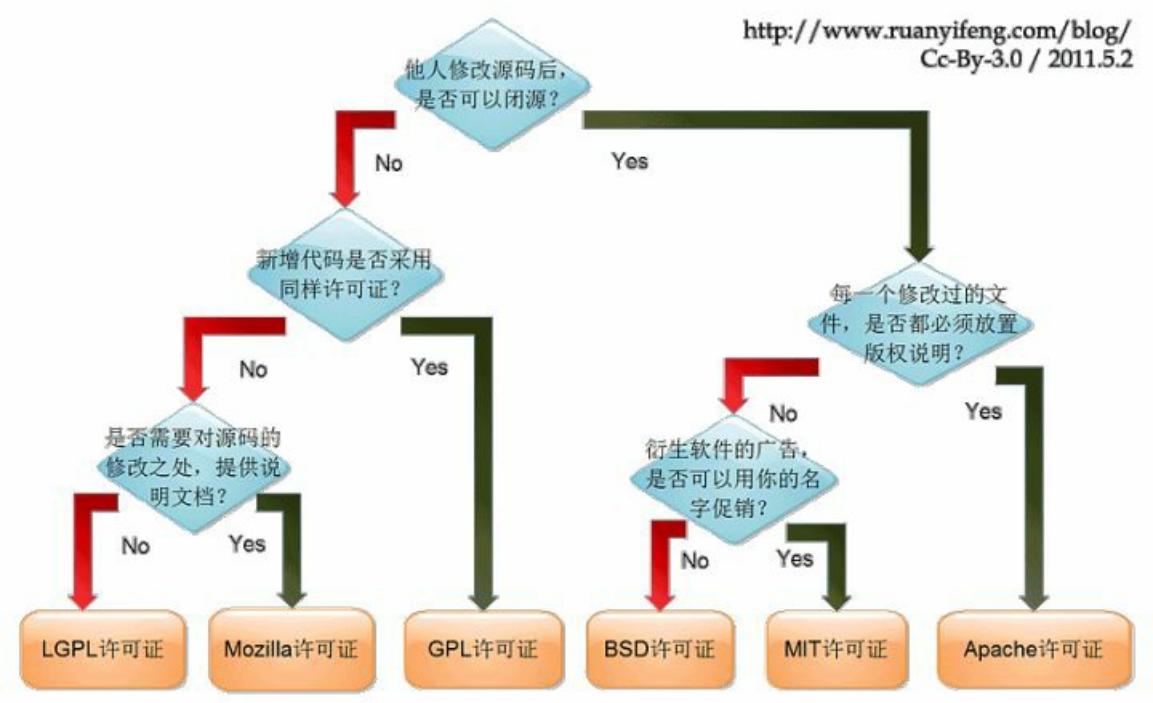
$\hat{f}\ \frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m}(y^i - h_\theta(x^i))x_j^i$

http://www.aleacubase.com/cudalab/cudalab_usage-math_formatting_on_markdown.html
https://www.zybuluo.com/codeep/note/163962#3%E5%A6%82%E4%BD%95%E8%BE%93%E5%85%A5%E6%8B%AC%E5%8F%B7%E5%92%8C%E5%88%86%E9%9A%94%E7%AC%A6

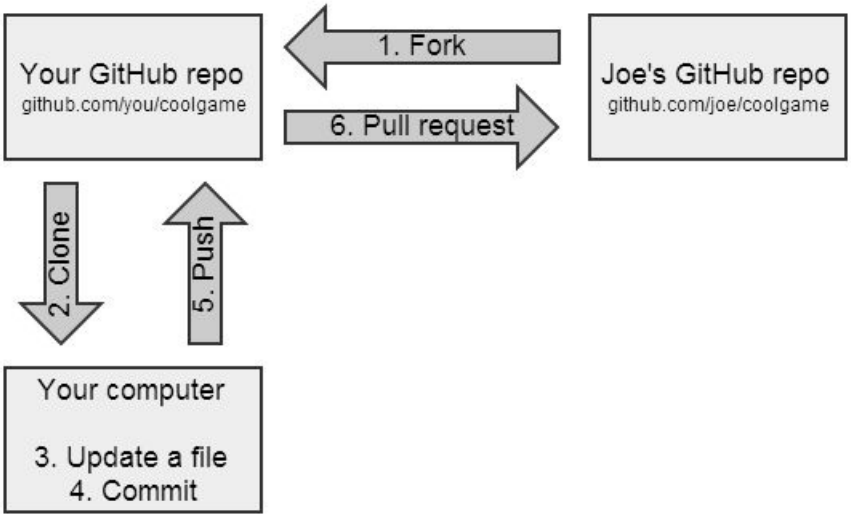
库函数的使用与说明

开源库的**使用**说明:

如何选择**开源**许**可**证？



Github Fork:



库函数地址

- [ ] C:\Data\Group\ShareFolder 是windows, linux的共享文件夹，有关Linux开发的文件都在这个文件夹，比如DeepLearning C:\Data\Group\ShareFolder\ToolsLibrary 存放的是Eigen, Opencv库
- [ ] https://github.com/tensorflow/models 里面有很多models，可以用来实际使用的。

Eigen

API document 整理一篇文章，阐述Eigen有哪些功能，能用来做啥。OPT++

数据集

https://archive.ics.uci.edu/ml/datasets.html

Google Test

Google C++ 单元测试框架 核心是添加include与lib路径，以及把运行时库设置成MTd 对工程名右键->属性->配置属性->C/C++->代码生成->运行时库：与前面gtest配置一样，选择MTd；代码位于：C:\Data\ShareFolder\Works\Miura\googletest-master

第五章:算法

第一节 Kriging插值

https://xg1990.com/blog/archives/222

空间插值问题，就是在已知空间上若干离散点 (x_i, y_i)的某一属性(如气温，海拔)的观测值

$z_i = z(x_i, y_i)$的条件下，估计空间上任意一点(x, y)的属性值的问题。
直观来讲，根据地理学第一定律，

大意就是，地理属性有空间相关性，相近的事物会更相似。由此人们发明了反距离插值，对于空间上任意一点(x, y) 的属性$z = z(x, y)$ ，

定义反距离插值公式估计量 $\hat{z} = \sum_{i=0}^{n} \frac{1}{d^{\alpha}} z_i$
其中α通常取1或者2。

即，用空间上所有已知点的数据加权求和来估计未知点的值，权重取决于距离的倒数(或者倒数的平方)。

那么，距离近的点，权重就大;距离远的点，权重就小。反距离插值可以有效的基于地理学第一定律估计属性值空间分布，但仍然存在很多问题:
α的值不确定 用倒数函数来描述空间关联程度不够准确

因此更加准确的克里金插值方法被提出来了

克里金插值公式$\hat{z_o} = \sum_{i=0}^{n} \lambda_i z_i$

其中$\hat{z_0}$是点(x_o, y_o)处的估计值，即:$z_o = z(x_o, y_o)$
假设条件:

1. 无偏约束条件 $E(\hat{z_0} - z_0) = 0$
2. 优化目标/代价函数 $J = Var((\hat{z_0} - z_0))$取极小值
3. 半方差函数r_{ij}与空间距离d_{ij}存在关联,并且这个关联可以通过这这两组数拟合出来，因此可以用距离d_{ij}来求得r_{ij}

半方差函数$r_{ij} = \sigma^2 - C_{ij}$; 等价于$r_{ij} = \frac{1}{2} E[(z_i - z_j)^2]$

其中:$C_{ij} = Cov(z_i, z_j) = Cov(R_i, R_j)$

求得r_{ij}之后，我们就可以求得λ_i

$$\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} & 1 \\ r_{21} & r_{22} & \cdots & r_{2n} & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{n1} & r_{n2} & \cdots & r_{nn} & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \cdots \\ \lambda_n \\ 0 \end{bmatrix} = \begin{bmatrix} r_{1o} \\ r_{2o} \\ \cdots \\ r_{no} \\ 1 \end{bmatrix}$$

最长递增子序列

对于序列5, 3, 4, 8, 6, 7, 其最长的递增子序列是3, 4, 6, 7,这里, 不需要保证元素是连在一起的, 只需要序号上升即可。问题变成求解n元系列a[1], a[2], a[3],...,a[n],

对于i < j < n, 假设我们已经求得前i个元素的最长递增子序列长度为MaxLen[i] 满足i < j, 那么我们可以遍历i从1到j-1来求得MaxLen[j]。基本的一些小算法写在一个CPP里面, 最后封装成一个类,一般要设计成模板类。

```
int Algo::LongestIncreaseSubsequence(int arr[], int arrSize)
{
    int *maxLen = new int[arrSize]();
    for (int i =0; i<arrSize; i++)
    {
        maxLen[i] = 1;
    }
    for (int j = 1; j < arrSize; j++)
    {
        int maxLenJ = 1;
        for (int i=0; i<j; i++)
        {
            if (arr[i] < arr[j])
            {
                maxLenJ = maxLen[i] + 1;
                maxLen[j] = std::max({ maxLen[j] ,maxLenJ });
            }
        }
    }
    return maxLen[arrSize-1];
}
```

LU分解

任何非奇异方阵都可以分解成上三角阵与下三角阵之积

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} * \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

对比两边矩阵的，可以求得：

但是编程时，行列都是从0开始时，要注意转换。

第1行：$a_{1j} = u_{1j}, j = 1, 2, .., n. => u_{1j} = a_{1j}$。

第1列：$a_{j1} = l_{j1}u_{11}, j = 1, 2, .., n. => l_{j1} = a_{j1}/u_{11}$。

...。

第k行：$a_{kj} = \sum_{i=1}^{i=k} l_{ki}u_{ij}, => u_{kj} = a_{kj} - \sum_{i=1}^{i=k-1} l_{ki}u_{ij}$。

第k列：$a_{jk} = \sum_{i=1}^{i=k} l_{ji}u_{ik}, => u_{jk} = [a_{jk} - \sum_{i=1}^{i=k-1} l_{ji}u_{ik}]/u_{kk}$。

因为前k-1行的u_{ij}都已知，前k-1列的l_{ij}都已知,因此可以求得第k行u_{ij}，第k列的l_{ij}。

问题：得保证a_{11}非0，以及矩阵非奇异。

利用LU分解求线性方程组的解

求解线性方程组 Ax=b相当于求解LUx=b;

设Y = UX; 因此LY = b;首先求解LY = b,

$$\begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{bmatrix}$$

求解上面的方程：

第1行：$y_1 = b_1$。

对于第k行：$b_k = \sum_{i=1}^{i=k} l_{ki}y_i => y_k = b_k - \sum_{i=1}^{i=k-1} l_{ki}y_i$。

求得Y之后，代入Y=UX求得X:

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}$$

对于上三角矩阵，我们从第n行开始求解

对第n行：$y_n = u_{nn}x_n => x_n = y_n/u_{nn}$。

...。

对第k行:$y_k = \sum_{i=k}^{i=n} u_{ki}x_i => x_k = [y_n - \sum_{i=k+1}^{i=n} u_{ki}x_i]/u_{kk}$
这样通过LU分解矩阵就求得了线性方程组的就X.

矩阵求逆

我们可以利用LU分解来求非奇异方阵的逆矩阵。
AB=I

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 1 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

可以分解成求:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} * \begin{bmatrix} b_{1k} \\ b_{2k} \\ \cdots \\ b_{nk} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 1 \\ \cdots \\ 0 \end{bmatrix}$$

右边的列，就只是第k行值非0；

$A * b_k = e_k$,对所有的e_k求出b_k就可以得到A的逆矩阵$A^{-1} = B$

实际求解中把A 换成LU来减少计算量。总的计算开销还是n^3。但是这样并不比直接的高斯消元法来的快。

算法实现如下。

```
//LU
template<class T>
void CMatrix<T>::LU(CMatrix &mat, int N, CMatrix* L, CMatrix* U)
{
    for (int k = 0; k <N; k++)
    {
        for (int j = k; j<N;j++)
        {
            T U_k_j = mat.Get(k, j);
            T L_j_k = mat.Get(j, k);
            for (int i = 0; i <k; i++)
            {
                U_k_j -= L->Get(k, i)*U->Get(i, j);
            }
            U->Set(k, j, U_k_j);

            for (int i = 0; i < k; i++)
            {
                L_j_k -= L->Get(j, i)*U->Get(i, k);
```

```
            }
            L_j_k = L_j_k / U->Get(k, k);
            L->Set(j, k, L_j_k);

        }
    }
}

//solve linear algebra equations
CVector Solve(CMatrix<double>& mat, CVector& vec)
{
    CVector X(vec.Size());
    CVector Y(vec.Size());
    if (mat.Columns() != mat.Rows() && mat.Rows() != vec.Size())
    {
        printf("Dimension not match!");
        return X;
    }
    CMatrix<double> L(vec.Size(), vec.Size());
    CMatrix<double> U(vec.Size(), vec.Size());
    mat.LU(mat, vec.Size(), &L, &U);
    Y = SolveLow(L, vec);
    cout << "Y\n" << Y;
    X = SolveUpper(U, Y);
    cout << "X\n" << X;
    return X;
}

CVector SolveLow(CMatrix<double>& mat, CVector& vec)
{
    CVector vecRes(vec.Size());
    if (mat.Columns() != mat.Rows() && mat.Rows() != vec.Size())
    {
        printf("Dimension not match!");
        return vecRes;
    }

    for (int k = 0; k < vec.Size(); k++)
    {
        double mRes = 0;
        mRes = vec.Get(k);
        for (int i = 0; i < k; i++)
        {
            mRes -= mat.Get(k, i)*vecRes.Get(i);
        }
        vecRes.Set(k, mRes);
    }
    return vecRes;
}

CVector SolveUpper(CMatrix<double>& mat, CVector& vec)
{
```

```
    CVector vecRes(vec.Size());
    if (mat.Columns() != mat.Rows() && mat.Rows() != vec.Size())
    {
        printf("Dimension not match!");
        return vecRes;
    }

    for (int k = vec.Size() -1; k >=0 ; k--)
    {
        double mRes = 0;
        mRes = vec.Get(k);
        for (int i = k+1; i <vec.Size(); i++)
        {
            mRes -= mat.Get(k, i)*vecRes.Get(i);
        }
        mRes = mRes / mat.Get(k, k);
        vecRes.Set(k, mRes);
    }
    return vecRes;
}

//inverse matrix
template<class T>
CMatrix<T>  CMatrix<T>::Inv()
{
    CMatrix result = *this;
    CMatrix<double> L(mRows, mColumns);
    CMatrix<double> U(mRows, mColumns);
    CMatrix<double> InvMat(mRows, mColumns);
    LU(result,mRows, &L, &U);
    for (int col = 0; col < mColumns; col++)
    {
        CVector vec(mRows, col);
        CVector mSolution(mRows);
        mSolution = SolveLow(L, vec);
        mSolution = SolveUpper(U, mSolution);
        for (int row = 0; row<mRows; row++)
        {
            InvMat.Set(row, col, mSolution.Get(row));
        }
    }
    return InvMat;
}
```

SVD

利用LU分解，我们可以求满秩的线性方程组的解。对于mxn（m>n）阶矩阵，对于超定方程（方程数目大于未知数的个数），因为一般没有解满足$Ax = b$,这就是最小二乘法发挥作用的地方。

这个问题的解就是使得：$||Ax - b||_2^2$值极小的解，通过求导（把x->x+e，求梯度等于0的x),可以得到解为：$x = (X^T A)^{-1} A^T b$

QR分解

定理：设A是mxn阶矩阵，m>=n,假设A为满秩的，则存在一个唯一的正交矩阵$Q(Q^T Q = I)$和唯一的具有正对角元$r_{ij} > 0$的nxn阶上三角阵R使得 $A = QR.$

Gram-Schmidt正交化

Gram-Schmidt正交化的基本想法，是利用投影原理在已有基的基础上构造一个新的正交基。

$((\beta))_1$

http://elsenaju.eu/Calculator/QR-decomposition.htm

https://rosettacode.org/wiki/QR_decomposition

https://www.wikiwand.com/en/QR_decomposition#/Example_2

https://www.wikiwand.com/en/Householder_transformation

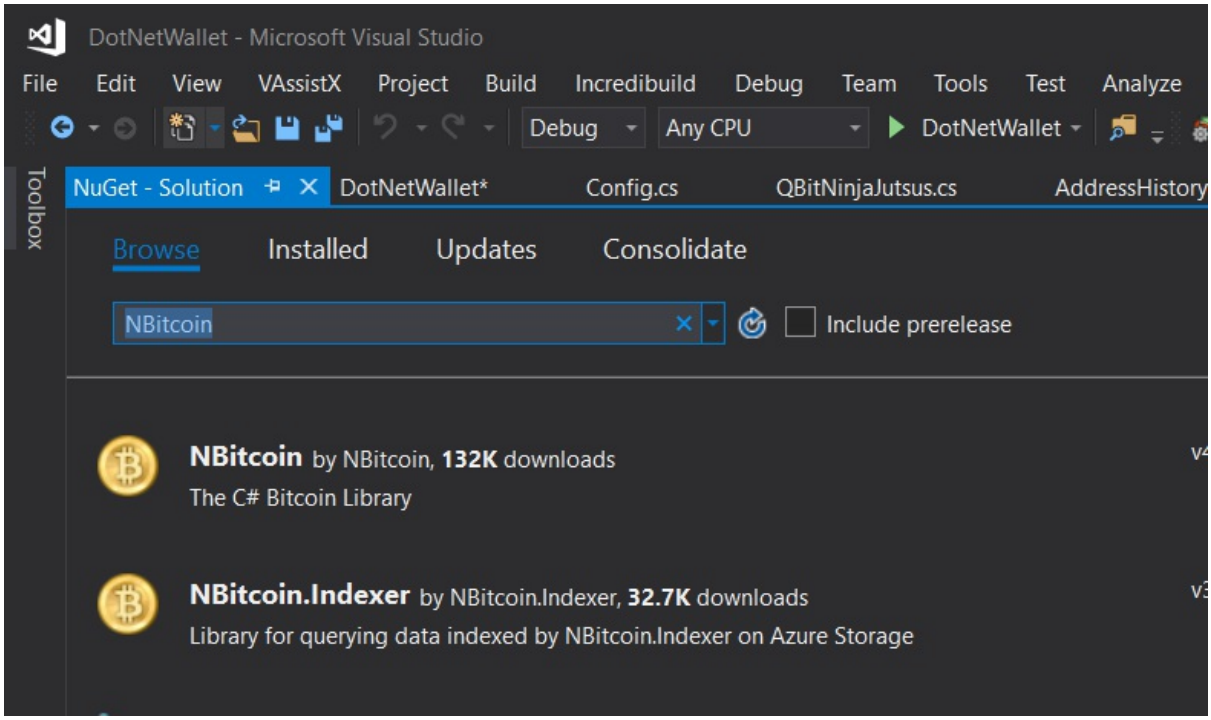
C#有开源的免费的代数库mathnet.numerics,功能也比较多，推荐通过Nuget来安装这个库

Nuget 安装dll

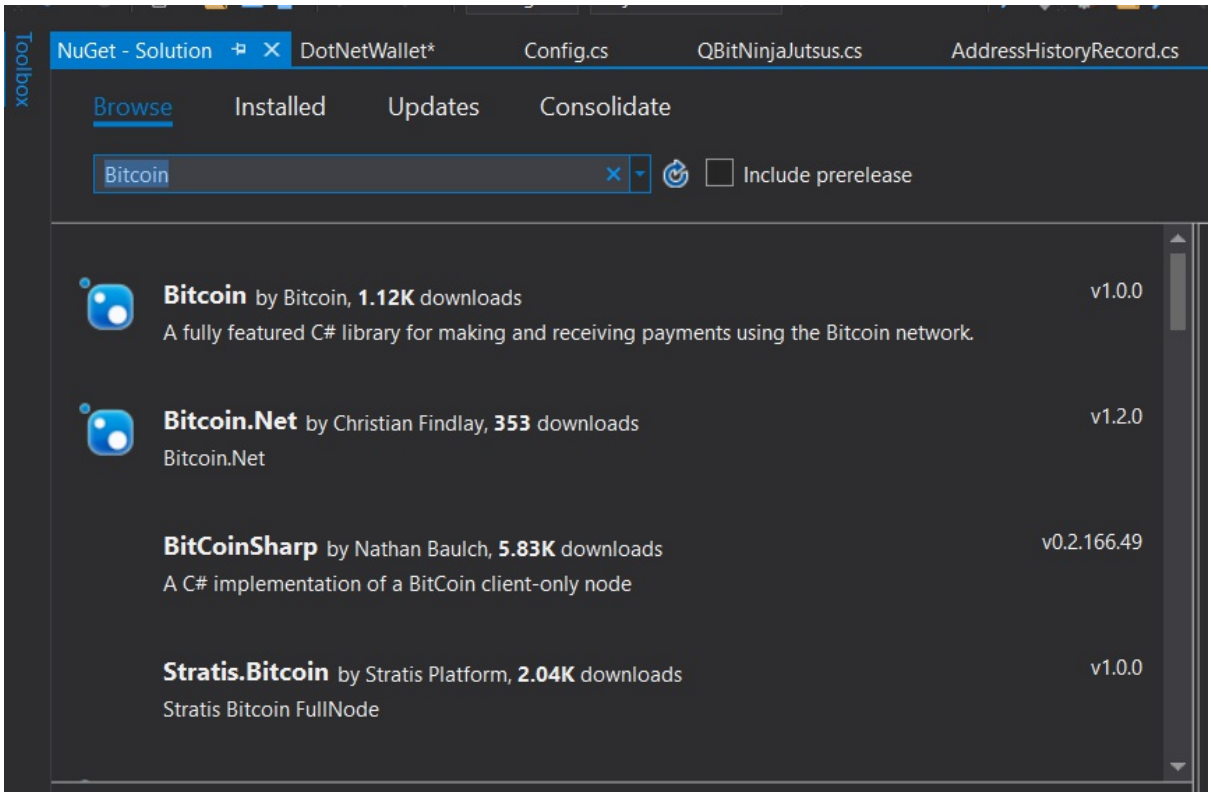
Tools-->Nuget Package Manager-->Manager Nuget packages for solution..-->Browse

https://numerics.mathdotnet.com/api/MathNet.Numerics.LinearAlgebra/Matrix`1.htm#QR

https://numerics.mathdotnet.com/matrix.html


DotNetWallet - Microsoft Visual Studio
File Edit View VAssistX Project Build Incredibuild Debug Team Tools Test Analyze
Debug
Any CPU
DotNetWallet
Toolbox
NuGet - Solution
DotNetWallet*
Config.cs
QBitNinjaJutsus.cs
AddressHistory
Browse
Installed
Updates
Consolidate
NBitcoin
Include prerelease
NBitcoin by NBitcoin, 132K downloads
The C# Bitcoin Library
NBitcoin.Indexer by NBitcoin.Indexer, 32.7K downloads
Library for querying data indexed by NBitcoin.Indexer on Azure Storage

研究这个


Toolbox
NuGet - Solution
DotNetWallet*
Config.cs
QBitNinjaJutsus.cs
AddressHistoryRecord.cs
Browse
Installed
Updates
Consolidate
Bitcoin
Include prerelease
Bitcoin by Bitcoin, 1.12K downloads
A fully featured C# library for making and receiving payments using the Bitcoin network.
v1.0.0
Bitcoin.Net by Christian Findlay, 353 downloads
Bitcoin.Net
v1.2.0
BitCoinSharp by Nathan Baulch, 5.83K downloads
A C# implementation of a BitCoin client-only node
v0.2.166.49
Stratis.Bitcoin by Stratis Platform, 2.04K downloads
Stratis Bitcoin FullNode
v1.0.0

医学图像重建算法

平行光束算法

2.6.4 FBP (先滤波后反投影) 算法的推导

我们首先给出二维傅里叶变换在极坐标系下的表达式

$$f(x,y)=\int_0^{2\pi}\int_0^{\infty}F_{polar}(\omega,\theta)e^{2\pi i\omega(x\cos\theta+y\sin\theta)}\omega d\omega d\theta \text{ 。}$$

因为 $F_{polar}(\omega,\theta)=F_{polar}(-\omega,\theta+\pi)$，所以

$$f(x,y)=\int_0^{\pi}\int_{-\infty}^{\infty}F_{polar}(\omega,\theta)|\omega|e^{2\pi i\omega(x\cos\theta+y\sin\theta)}d\omega d\theta \text{ 。}$$

根据中心切片定理，我们可以用 P 来代替 F：

$$f(x,y)=\int_0^{\pi}\int_{-\infty}^{\infty}P(\omega,\theta)|\omega|e^{2\pi i\omega(x\cos\theta+y\sin\theta)}d\omega d\theta \text{ 。}$$

我们意识到 $|\omega|$ 是斜坡率波器的传递函数，令 $Q(\omega,\theta)=|\omega|P(\omega,\theta)$，则

$$f(x,y)=\int_0^{\pi}\int_{-\infty}^{\infty}Q(\omega,\theta)e^{2\pi i\omega(x\cos\theta+y\sin\theta)}d\omega d\theta \text{ 。}$$

利用一维傅里叶反变换并记 Q 的反变换为 q，我们最后得到

$$f(x,y)=\int_0^{\pi}q(x\cos\theta+y\sin\theta,\theta)d\theta \text{ ，}$$

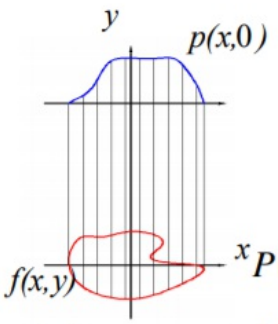
或

$$f(x,y)=\int_0^{\pi}q(s,\theta)|_{s=x\cos\theta+y\sin\theta}\,d\theta \text{ 。}$$

这正是 $q(s,\theta)$ 的反投影 (见第1.5节)。

中心切片定理

Special example prove



1. 平行于y轴投影

$$p(x,0)=\int_{-\infty}^{+\infty} f(x,y)dy \qquad (1)$$

2. 投影函数一维傅里叶变换

$$P(u)=\int_{-\infty}^{+\infty} p(x,0)e^{-j2\pi ux}dx=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-j2\pi ux}dxdy \qquad (2)$$

3. 密度函数二维傅里叶变换

$$F(u,v)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-j2\pi(ux+vy)}dxdy \qquad (3)$$

4. 在v=0时，傅里叶变换表示

$$F(u,v)\big|_{v=0}=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-j2\pi ux}dxdy \qquad (4)$$

这表明，
一个物体0°投影的傅里叶变换与该物体二维傅里叶变换中v=0的直线相同。

5. 式（2）与（4）相等，证明完毕

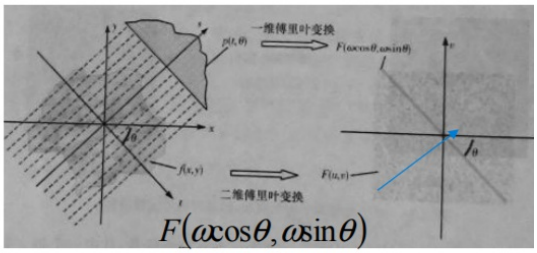
3. 对投影函数中的变量t进行傅里叶变换

$$P(\omega,\theta)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f'(t,s)dse^{-j2\pi\omega t}dt \qquad (3)$$

4. 对（3）式右面进行坐标变换x，y

$$dsdt=Jdxdy=\begin{vmatrix}\partial t/\partial x & \partial s/\partial x\\ \partial t/\partial y & \partial s/\partial y\end{vmatrix}dxdy=dxdy$$

雅克比转换



傅里叶切片定理：
物体f(x,y)平行投影的傅里叶变换，是物体二维傅里叶变换的一个切片（直线），切片是在与投影相同的角度获得的。

$$P(\omega,\theta)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-j2\pi\omega(x\cos\theta+y\sin\theta)}dxdy \qquad (4)$$

5. f(x,y)二维傅里叶变换

$$F(u,v)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-j2\pi(ux+vy)}dxdy \qquad (5)$$

6. 比较公式（4）和（5），令 $u=\omega\cos\theta, v=\omega\sin\theta$

$$F(\omega\cos\theta,\omega\sin\theta)=P(\omega,\theta) \qquad (6)$$

平行光束算法到扇形光束算法

其本质就是从直角坐标系变换到极坐标系

在完成了从平行光束数据 $p(s,\theta)$ 到扇形束数据 $g(\gamma,\beta)$ 的替换，旧变量 s 和 θ 到新变量 γ 和 β 的替换，及加入一个雅可比因子 $J(\gamma,\beta)$，一个崭新的扇形束图像重建算法就诞生了 (图 3.5)!

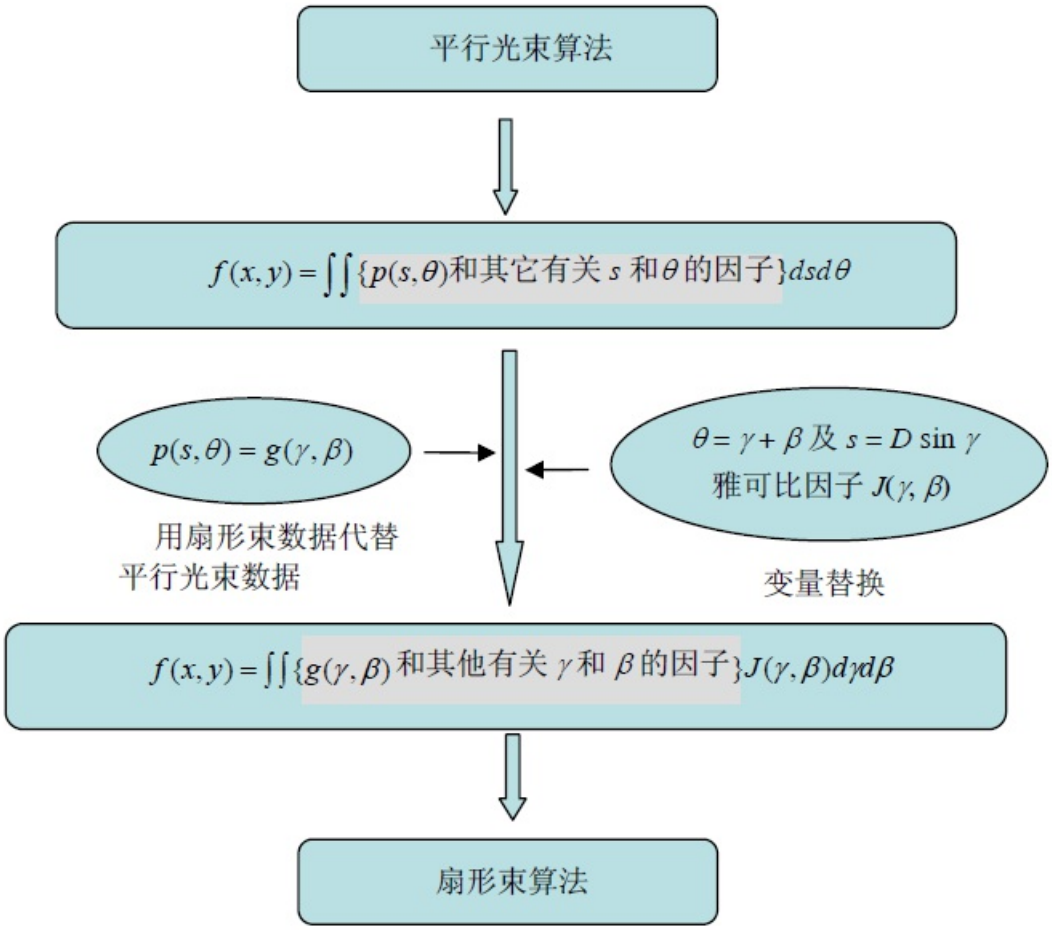


图 3.5 从平行光束图像重建算法到扇形束图像重建算法的推导过程。

投影

1.5.1 投影

设 $f(x, y)$ 为 x-y 平面上定义的密度函数，其投影函数(即射线和，线积分，及 拉东变换) $p(s, \theta)$ 有下面不同的等价表达式：

$$p(s,\theta)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(x,y)\delta(x\cos\theta+y\sin\theta-s)dxdy\,,$$

$$p(s,\theta)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(x,y)\delta(\bar{x}\cdot\bar{\theta}-s)dxdy\,,$$

$$p(s,\theta)=\int_{-\infty}^{\infty}f(s\cos\theta-t\sin\theta,s\sin\theta+t\cos\theta)dt\,,$$

$$p(s,\theta)=\int_{-\infty}^{\infty}f(s\bar{\theta}+t\bar{\theta}^{\perp})dt\,,$$

$$p(s,\theta)=\int_{-\infty}^{\infty}f_{\theta}(s,t)dt\,,$$

其中 $\bar{x}=(x,y)$， $\bar{\theta}=(\cos\theta,\sin\theta)$， $\bar{\theta}^{\perp}=(-\sin\theta,\cos\theta)$，$\delta$是狄拉克$\delta$函数，图像 f 旋转角度 θ 后记为 f_{θ}。我们假设探测器按逆时针方向绕物体旋转。这等价于探测器不动，而物体按顺时针做旋转. 有关的坐标系如图1.12所示。

狄拉克函数

$$\int_{-\infty}^{\infty}\delta(ax)f(x)dx=\frac{1}{|a|}f(0)\,,$$

$$\int_{-\infty}^{\infty}\delta^{(n)}(x)f(x)dx=(-1)^n f^{(n)}(0)\ \text{[第 } n \text{ 阶导数]},$$

$\delta(g(x))f(x)=\sum_n\frac{1}{|g'(\lambda_n)|}\delta(x-\lambda_n)$ ，其中 λ_n 为 $g(x)$ 的零点。

狄拉克δ函数在二维和三维的情形的定义分别是， $\delta(\bar{x})=\delta(x)\delta(y)$ 和 $\delta(\bar{x})=\delta(x)\delta(y)\delta(z)$ 。这时，在上面的最后一个性质中， $|g'|$ 要分别被 $|grad(g)|=\sqrt{\left(\frac{\partial g}{\partial x}\right)^2+\left(\frac{\partial g}{\partial y}\right)^2}$ 和 $|grad(g)|=\sqrt{\left(\frac{\partial g}{\partial x}\right)^2+\left(\frac{\partial g}{\partial y}\right)^2+\left(\frac{\partial g}{\partial z}\right)^2}$ 取代。

在二维成像中，我们通常用δ函数 $\delta(\bar{x}-\bar{x}_0)$ 来表示位于 $\bar{x}=\bar{x}_0$ 的点源。函数 $f(\bar{x})=\delta(\bar{x}-\bar{x}_0)=\delta(x-x_0)\delta(y-y_0)$ 的拉东变换就是

$$p(s,\theta)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(\bar{x})\delta(\bar{x}\cdot\bar{\theta}-s)d\bar{x}\,,$$

$$p(s,\theta)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\delta(x-x_0)\delta(y-y_0)\delta(x\cos\theta+y\sin\theta-s)dxdy\,,$$

解析法应用的就是中心切片定理，迭代法一般应用共轭梯度法或者拟牛顿法。

My Awesome Book

This file file serves as your book's preface, a great place to describe your book's content and ideas.

Levenberg-Marquardt算法

Levenberg-Marquardt

方法用于求解非线性最小二乘问题，结合了梯度下降法和高斯-牛顿法。

其主要改变是在Hessian阵中加入对角线项,加这一样可以保证Hassian阵是正定的。当然，这是一种L2正则化方式。

- $x_{t+1} = x_t - (H + \lambda I_n)^{-1} g,$
- $x_{t+1} = x_t - (J^T J + \lambda I_n)^{-1} J^T r$

L-M算法的不足点。

当λ很大时，$J^T J + \lambda I_n$根本没用，Marquardt为了让梯度小的方向移动的快一些，来防止小梯度方向的收敛，把中间的单位矩阵换成了$diag(J^T J)$,因此迭代变成：

- $x_{t+1} = x_t - (J^T J + \lambda diag(J^T J))^{-1} J^T r$

阻尼项(damping parameter)λ的选择，Marquardt 推荐一个初始值λ_0与因子$v > 1$，开始时，

$\lambda = \lambda_0$,然后计算cost functions,第二次计算$\lambda = \lambda_0 / v$,如果两者cost function都比初始点高，然后我们就增大阻尼项，通过乘以v,直到我们发现当$\lambda = \lambda_0 v^k$时，cost function下降。

如果使用$\lambda = \lambda_0 / v$使得cost fucntion下降，然后我们就把λ_0 / v作为新的λ

当$\lambda = 0$时，就是高斯-牛顿法，当λ趋于无穷时，就是梯度下降法。如果使用λ / v没有是损失函数下降，使用λ导致损失函数下降，那么我们就继续使用λ做为阻尼项。

$$
\begin{aligned}
&k := 0; x := x_0; v := 2; A := J(x)^T J(x) g := J(x)^T f(x) \\
&u := \tau * max(a_{ii}) \\
&found := (||g||_\infty \le \epsilon_1) \\
&while(not found) and (k < k_{max}) \\
&k := k + 1; solve- > (A + uI) h_{lm} = -g \\
&if \quad ||h_{lm}|| \le \epsilon(||x|| + \epsilon_2) \\
&found := true \\
&else: \\
&x_{new} := x + h_{lm} \\
&\rho := (F(x) - F(x + h_{dl}) / (L(0) - L(h_{dl}))) \\
&if \quad \rho \ge 0 \\
&x := x_{new}; g := J(x)^T f(x) \\
&A := J(x)^T J(x) g := J(x)^T f(x) \\
&found := (||g||_\infty \le \epsilon_1) \\
&u := u * max(1/3, 1 - (2\rho - 1)^3); v := 2 \\
&else
\end{aligned}
$$

```
    def LM_Solver(self):
        init_p = 0.5 * np.ones((2, 1))
```

```
        init_p[0, 0] = 2.0
        init_p[1, 0] = 1.0
        epsilon_1 = 1e-15
        epsilon_2 = 1e-15
        epsilon_3 = 1e-15
        alpha = 1
        cost_history = np.zeros((self.training_steps, 3))
        Jaco = self.Jacobian(init_p)
        Hassian = np.dot(Jaco.transpose(), Jaco)
        resi = self.residual(init_p)
        m_g = np.dot(Jaco.transpose(), resi)
        cost_old = np.dot(resi.transpose(), resi)
        epoch = -1
        train_stop = -1
        miu = Hassian.max()
        lamda_v = 2

        matrix_I = np.identity(Hassian.shape[0])
        while epoch < self.training_steps - 1 and train_stop < 0:
            epoch += 1
            print "Epoch: ", epoch
            go_next_epoch = -1

            while go_next_epoch < 0:
                sigma_p = np.dot(np.linalg.inv(Hassian + miu*matrix_I), m_g)
                square_g = np.linalg.norm(sigma_p)
                if square_g <= epsilon_2*np.linalg.norm(init_p):
                    train_stop = 1
                    break
                else:
                    new_p = init_p + sigma_p
                    resi_new = self.residual(new_p)
                    cost_new = np.dot(resi_new.transpose(), resi_new)
                    frac = np.dot(sigma_p.transpose(),  miu* sigma_p + m_g)
                    rou = (cost_old - cost_new) / frac
                    if rou > 0:
                        go_next_epoch = 1
                        init_p = new_p
                        Jaco = self.Jacobian(init_p)
                        Hassian = np.dot(Jaco.transpose(), Jaco)
                        resi = self.residual(init_p)
                        m_g = np.dot(Jaco.transpose(), resi)
                        if (np.abs(m_g)).max < epsilon_1:
                            train_stop = 1
                            break
                        miu *= max(1.0 / 3.0, 1 - pow(2 * rou - 1, 3))
                        lamda_v = 2
                        print init_p.transpose(), miu, cost_new
                        cost_history[epoch, 0] = cost_new
                        cost_history[epoch, 1] = miu
                        cost_history[epoch, 2] = np.linalg.norm(m_g)
                        cost_old = cost_new
```

```
                else:
                    miu *= lamda_v
                    # lamda_miu *= 0
                    lamda_v *= 2
        return cost_history, init_p
```

线性搜索与Armijo准则

符号约定:

- g_k : $\nabla f(x_k)$,即目标函数关于k次迭代值x_k的导数
- G_k : $G(x_k) = \nabla^2 f(x_k)$,即Hassian矩阵
- d_k :第k次迭代的步长因子，在最速下降算法中，有$d_k = -g_k$
- α_k:第k次迭代的步长因子，有$x_{k+1} = x_k + \alpha_k d_k$

在精确线性搜索中，步长因子α_k由下面的因子确定:

- $\alpha_k = argmin_\alpha f(x_k + \alpha d_k)$

而对于非精确线性搜索，选取的α_k只要使得目标函数f得到可接受的下降量，即:

- $\Delta f(x_k) = f(x_k) - f(x_k + \alpha_k d_k)$

Armijo 准则用于非精确线性搜索中步长因子α的确定，内容如下:
Armijo 准则:

已知当前位置x_k和优化方向d_k，参数$\beta \in (0,1), \delta \in (0, 0.5)$.令步长因子 - $\alpha_k = \beta^{m_k}$,其中

m_k为满足下列不等式的最小非负整数m:

- $f(x_k + \beta^m d_k) \leq f(x_k) + \delta \beta^m g_k^T d_k$

由此确定下一个位置$x_{k+1} = x_k + \alpha_k d_k$
对于梯度上升，上面的方程变成:

- $f(x_k - \beta^m d_k) \geq f(x_k) - \delta \beta^m g_k^T d_k$

由此确定下一个位置$x_{k+1} = x_k - \alpha_k d_k$

Qusi-Netwon methods

BFGS

我们知道，在牛顿法中，我们需要求解二阶导数矩阵--Hassian阵，当变量很多时，求解Hassian阵势比较费时间的，Qusi-Netwon法主要是在构造Hassian阵上下功夫，它是通过构造一个近似的Hassian阵，或者Hassian阵的逆，而不是解析求解或者利用差分法来求解这个Hassian阵。构造的Hassian阵通过迭代而改变。

比较出名的Qusi-Netwon方法有BFGS(以Charles George Broyden, Roger Fletcher, Donald Goldfarb and David Shanno命名)

在牛顿法中，k步搜寻步长与方向是p_k，满足下面方程

- $B_k p_k = -\nabla f(x_k)$

- B_k就是近似的Hassian 阵。下面我们讨论B_k如何变化，

我们要求B_k的更新满足quasi-Netwon条件

- $B_{k+1}(x_{k+1} - x_k) = \nabla f(x_{K+1} - \nabla f(x_K)$

这个条件就是简单的求$f(x)$的二阶导数。
令：

- $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$，$s_k = x_{k+1} - x_k$，因此
- B_{k+1}满足$B_{k+1}s_k = y_k$

这就是割线方程（the secant equation），The curvature condition $s_k^T y_k > 0$需要满足。
k步的Hassian阵以如下方式更新，

- $B_{k+1} = B_k + U_k + V_k$

为了保持B_{k+1}的正定性以及对称性。B_{k+1}可以取如下形式：

- $B_{k+1} = B_k + \alpha u u^T + \beta v v^T$

选择$u = y_k$,$v = B_k s_k$,为了满足割线方程（the secant condition），我们得到：

- $\alpha = \frac{1}{y_k^T s_k}$
- $\beta = \frac{1}{s_k^T B_k s_k}$

最终我们得到Hassian阵的更新方程：

- $B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k^T}{s_k^T B_k s_k}$

利用 Sherman–Morrison formula，

- $(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}$

其中A是可逆方阵，$1 + v^T A^{-1} u \neq 0$

可以方便的得到B_{k+1}的逆。

- $B_{k+1}^{-1} = (I - \frac{s_k y_k^T}{y_k^T s_k})B_k^{-1}(1 - \frac{y_k s_k^T}{y_k^T s_k}) + \frac{s_k s_k^T}{y_k^T s_k}$
- $B_{k+1}^{-1} = B_k^{-1} + \frac{(s_k^T y_k + y_k^T B_k^{-1} y_k)(s_k s_k^T)}{(s_k^T y_k)^2} - \frac{B_k^{-1} y_k s_k^T + s_k y_k^T B_k^{-1}}{s_k^T y_k}$

DFP

参考 DFP的矫正公式如下：

$$H_{k+1} = H_k + \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{s_k^T y_k}$$

当采用精确线搜索时，矩阵序列H_k的正定新条件$s_k^T y_k > 0$可以被满足。但对于Armijo搜索准则来说，不能满足这一条件，需要做如下修正：

$$H_{k+1} = H_k \quad s_k^T y_k \leq 0$$

$$H_{k+1} = H_k + \frac{H_k y_k {y_k}^T H_k}{y\_k^T H\_k y\_k} \quad s_k^T y_k > 0$$

Broyden族算法

之前BFGS和DFP矫正都是由y_k和$B_k s_k$(或者s_k和$H_k y_k$组成的秩2矩阵。而Droyden族算法采用了BFGS和DFP校正公式的凸组合，如下：

$$H_{k+1}^{\phi} = \phi_k H_{k+1}^{BFGS} + (1-\phi_k) H_{k+1}^{DFP} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{s_k^T y_k} + \phi_k v_k v_k^T$$ 其中

$\phi_k \in [0,1]$，v_k由下式定义：

$$v_k = \sqrt{y_k^T H_k y_k}\left(\frac{s_k}{y_k^T s_k} - \frac{H_k y_k}{y_k^T H_k y_k}\right)$$

Gauss-Netwon方法

对于函数$f(x)$的小邻域展开

- $f(x+h) \simeq l(h) \equiv f(x) + J(x)h$
- $F(x+h) \simeq L(h) \equiv \frac{1}{2}l(h)^T l(h) = \frac{1}{2}f^T f + h^T J^T f + \frac{1}{2}h^T J^T J h$

我们需要选择步长h_{gn}来最小化$L(h)$

- $h_{gn} = argmin_h L(h)$
- $(J^T J)h_{gn} = -J^T f$

信赖域方法

Powell's Dog Leg Method是一种信赖域方法

在Gauss-Netwon迭代中

- $J(x)h \simeq -f(x)$

最陡的方向由下面公式给出:

- $h_{sd} = -g = -J(x)^T f(x)$

但是这只是给出了方向，而没有给出步长。

考虑线性模型

- $f(x+\alpha h_{sd}) \simeq f(x) + \alpha J(x) h_{sd}$
- $F(x+\alpha h_{sd}) \simeq \frac{1}{2}||f(x) + \alpha J(x) h_{sd}||^2 = F(x) + \alpha h_{sd}^T J(x)^T f(x) + \frac{1}{2}\alpha^2 ||J(x)h_{sd}||^2$

当α取如下值得时候，以上函数取最小值

- $\alpha = -\frac{h_{sd}^T J(x)^T f(x)}{||J(x)h_{sd}||^2} = \frac{||g||^2}{||J(x)h_{sd}||^2}$

现在有两个步长的选择 $a = \alpha h_{sd}$ 以及 $b = h_{gn}$,Powell建议在信赖域半价是Δ的时候，步长可以如下选择

If $||h_{gn}|| \leq \Delta, h_{dl} = h_{gn}$

- Elseif: $||\alpha h_{sd}|| \geq \Delta, h_{dl} = (\Delta / ||h_{sd}||) h_{sd}$

else:

- $h_{dl} = \alpha h_{sd} + \beta(h_{gn} - \alpha h_{sd})$

选择β 使得$||h_{dl}|| = \Delta$

在L-M算法中我们定义了增益因子，

- $\rho = (F(x) - F(x+h_{dl}) / (L(0) - L(h_{dl})))$

其中L是线性模型

- $L(h) = \frac{1}{2}||f(x) + J(x)h||^2$

在L-M我们通过ρ来控制阻尼因子，在dog-leg算法中，我们通过它来控制步长

Dog Leg Method

$$
\begin{aligned}
&k := 0; x := x_0; \Delta := \Delta_0; g := J(x)^T f(x) \\
&found := (||f(x)||_\infty \le \epsilon_3) or (||g||_\infty \le \epsilon_1) \\
&while (not found) and (k < k_{max}) \\
&k := k + 1; computer \alpha by (2.4) \\
&h_{sd} := -\alpha g; solve J(x) h_{gn} \simeq -f(x) \\
&computer h_{dl} by (4.5) \\
&if ||h_{dl}|| \le \epsilon(||x|| + \epsilon_2) \\
&found := true \\
&else : \\
&x_{new} := x + h_{dl} \\
&\rho := (F(x) - F(x + h_{dl})/(L(0) - L(h_{dl}))) \\
&if \quad \rho \ge 0 \\
&x := x_{new}; g := J(x)^T f(x) \\
&found := (||f(x)||_\infty \le \epsilon_3) or (||g||_\infty \le \epsilon_1) \\
&if \quad \rho \ge 0.75 \\
&\Delta := max(\Delta, 3 * ||h_{dl}||) \\
&elseif \quad \rho < 0.25 \\
&\Delta := \Delta/2; found := (\Delta \le \epsilon_2(||x|| + \epsilon)
\end{aligned}
$$

```
    def Dogleg_Solver(self):
        init_para = np.ones((2, 1))
        epsilon_1 = 1e-15
        epsilon_2 = 1e-15
        epsilon_3 = 1e-15
        alpha = 1
        cost_history = np.zeros((self.training_steps, 3))
        Jaco = self.Jacobian(init_para)  //get Jacobian
        Hassian = np.dot(Jaco.transpose(), Jaco)
        resi = self.residual(init_para)
        m_g = np.dot(Jaco.transpose(), resi)
        cost_old = np.dot(resi.transpose(), resi)


        epoch = -1
        train_stop = -1
        delta = 0.01
        while epoch < self.training_steps - 1 and train_stop < 0:
            epoch += 1
            print "Epoch: ", epoch
            go_next_epoch = -1

            while go_next_epoch < 0:
                square_g = np.linalg.norm(m_g)
                square_Jg = np.linalg.norm(np.dot(Jaco, m_g))
                alpha = pow(square_g, 2) / pow(square_Jg, 2)
```

```
                h_sd = -alpha * m_g
                h_gn = np.dot(np.linalg.inv(Hassian), -m_g)
                norm_para = np.linalg.norm(init_para)

                ##calculate h_dl
                if np.linalg.norm(h_gn) <= delta:
                    h_dl = h_gn
                    hdl_type = 0
                elif np.linalg.norm(h_sd) >= delta:
                    h_dl = (delta / np.linalg.norm(h_sd)) * h_sd
                    hdl_type = 1
                else:
                    beta = self._get_h_dl(h_sd, h_gn, delta)
                    h_dl = h_sd + beta * (h_gn - h_sd)
                    hdl_type = 2

                    # #epsilon_2*norm_para:
                if np.linalg.norm(h_dl) < epsilon_2 * (norm_para + epsilon_2):
                    train_stop = 1
                    break
                else:
                    para_0 = init_para + h_dl
                    #cost_old = pow(np.linalg.norm(resi), 2)
                    resi_new = self.residual(para_0)
                    cost_new = np.dot(resi_new.transpose(), resi_new)
                    #pow(np.linalg.norm(resi_new), 2)
                    #h_sub = h_sd - 0.5 * np.dot(Hassian, h_dl)
                    #frac = np.dot(h_dl.transpose(), h_sub)
                    if hdl_type == 0:
                        frac = cost_new
                    elif hdl_type == 1:
                        frac = delta/(2*alpha)*(2*np.linalg.norm(alpha*m_g) - delta)
                    else:
                        frac = 0.5*alpha*pow(beta, 2)*pow(np.linalg.norm(alpha*m_g), 2)
                        frac += beta*(2 - beta)*cost_new
                    rou = (cost_old - cost_new) /frac
                    if rou > 0:
                        go_next_epoch = 1
                        init_para = para_0
                        Jaco = self.Jacobian(init_para)
                        Hassian = np.dot(Jaco.transpose(), Jaco)
                        resi = self.residual(init_para)
                        m_g = np.dot(Jaco.transpose(), resi)

                        cost_history[epoch, 0] = cost_new
                        cost_history[epoch, 1] = delta
                        cost_history[epoch, 2] = np.linalg.norm(m_g)
                        print para_0.transpose(), cost_new, delta

                        if (np.abs(m_g)).max() < epsilon_1 or (np.abs(resi_new)).max() < epsi
lon_3:
                            train_stop = 1
```

```
                        break
                if rou > 0.75:
                    delta = max(delta, 3 * np.linalg.norm(h_dl))
                elif rou < 0.25:
                    delta /= 2.0
                    if delta < epsilon_2:
                        break
                cost_old = cost_new
        return cost_history, init_para
```

优化收敛位置

参考知乎上面

对于N个参数的系统(神经网络)，对应的Hassian阵是N阶的，我们假定其本征值的正负概率都是0.5.因此，要保证是局部最小值，则Hassian正定，意味着所有的本征值都为正。其概率是$\frac{1}{2^N}$。同样，是局部最大值，则Hassian阵负定，概率也为$\frac{1}{2^N}$。因此，最有可能的是本征值有正有负，也就是鞍点的情形。但是即使是普通的鞍点，函数不会震荡，而是马上逃离。

实际的情形是，函数收敛到了一个大的平坦区域，其表现就是一阶导数很小，以至于在训练结束的时候还没有逃离出这个平坦区域。那么，我们得从数学上讨论这种平坦区域(平坦马鞍面)具有什么样的性质。可以看下面一个方程：

$f(x,y)=\frac{\alpha}{2}x^2-\frac{\beta}{2}y^2$

$f_x=\alpha x; f_y=-\beta y$.假设$\alpha>0,\beta>0$

(0, 0)点是唯一的鞍点。在x方向，该点是局部最小值，但在y方向，是局部极大值，如果β很小，很小是它与学习率r相乘结果与1来说的。假设学习率是r一开始$y=\Delta_0$,则一次迭代后

$\Delta_1=\Delta_0+r*\beta*\Delta_0=(1+r\beta)\Delta$

$\Delta_2=\Delta_1+r*\beta*\Delta_1=(1+r\beta)\Delta_1$

n次迭代后：

如果训练总的次数为N，$r\beta\leq\frac{1}{N}$,那N次迭代之后

$(1+r\beta)^N\Delta_0\leq(1+\frac{1}{N})^N\approx e\Delta_0=2.71828\Delta_0$，这定量的给出了在训练结束的时候，函数能多大程度的逃离(0，0)点。

这个模型可以推广到N维情形。假设Hassian阵有N-K个本征值非负，K个小于零。

考虑通过平移后，鞍点处于(0，0....0)点附近的二阶展开：

$L(X)=L(0)+\sum_{i=K+1}^{i=N}\frac{\alpha_i}{2}x_i^2-\sum_{i=1}^{i=K}\frac{\alpha_i}{2}x_i^2$

如果满足所有的$\alpha_i\ll1,i\in[1,K]$，这时候，函数会在很长时间内，局域在鞍点附件。

可以考虑的时，怎么通过设计Cost Function来避免产生这些超级马鞍面。

这些超级马鞍面的产生要考虑激活函数吗，时候Relu比SIgmoid不容易产生这些马鞍面？

激活函数的作用

增加非线性，增强学习能力

为啥使用收敛慢的(随机)梯度下降法

因为只需要计算一阶导数，而不需要计算二阶，因为几十万以上的参数，计算二劫导数太费时间，内存也是个问题(除非用L-BFGS)

SVD PCA

SVD 将矩阵分解成累加求和的形式，其中每一项的系数即是原矩阵的奇异值。这些奇异值，按之前的几何解释，实际上就是空间超椭球各短轴的长度。现在想象二维平面中一个非常扁的椭圆(离心率非常高)，它的长轴远远长于短轴，以至于整个椭圆看起来和一条线段没有什么区别。这时候，如果将椭圆的短轴强行置为零，从直观上看，椭圆退化为线段的过程并不突兀。回到 SVD 分解当中，较大的奇异值反映了矩阵本身的主要特征和信息;较小的奇异值则如例中椭圆非常短的短轴，几乎没有体现矩阵的特征和携带的信息。因此，若我们将 SVD 分解中较小的奇异值强行置为零，则相当于丢弃了矩阵中不重要的一部分信息。

因此，SVD 分解至少有两方面作用:

- 分析了解原矩阵的主要特征和携带的信息(取若干最大的奇异值)，这引出了主成分分析(PCA);
- 丢弃忽略原矩阵的次要特征和携带的次要信息(丢弃若干较小的奇异值)，这引出了信息有损压缩、矩阵低秩近似等话题。

项目背景

随着半导体工艺中，芯片中的尺寸越来越小，光学衍射效应越来越明显，单纯的几何光学（初中物理老师告诉你，光是直线传播的，这其实说的是，光是粒子，但光通过狭窄的缝隙会发生衍射效应，这时，光不是直线传播的了）以及不适用，要考虑光的衍射效应。光通过Musk就相当于一次傅里叶变化，从时域变到了频率域。再通过棱镜，又相当于一次傅里叶变换，从频率域变到时域。但棱镜的尺寸有限，因此只接收了低频波，因此，用图像处理的话来说，棱镜的作用就是一个低通滤波器。去掉了高频部分，因此投影到wafer（晶圆）上的图案不再菱角分明，而是在边界处会比较圆滑。

如果没有一些傅里叶光学的背景，你们听起来会比较费劲，但是如果你们有图像处理的背景的话，就记住把Musk上的图案刻蚀到晶圆上就是经历了两次傅里叶变换再加一个低通滤波器。

光通过单缝后，其频谱是连续的，但是经过周期性（结构）缝，其频谱就是离散的，如果为了节约棱镜的开支，就只能取很低的频谱。

光学建模是个逆问题（逆问题的例子，就是CT，通过收集到的信号来反推身体的内部结构）。（怎么把逆问题与深度学习，机器学习联系起来？），再通俗点，它实际上和机器学习，深度学习要处理的问题是一样的，就是求解一个模型，就是train一个model。正向过程就是你已经训练好了一个模型（取啥例子？），然后输入一张图，看它的分类是啥，这很容易，难得是，我有一堆分类好的图片，让你训练一个model。逆问题就是一个优化问题，因此就设计到一些优化算法，这时候考点就来了？有哪些优化算法，总结下，有两种，线性搜索方法（linear search）以及 trust region。但是对于这个求逆问题而言，是很复杂的，直接离散的话，参数基本在10的15次方以上，总之，这个求逆问题是很难的，即使求出来，也不会很准。因此，我们有两个研究的方向。直接通过光学图像（也就是将要刻蚀在wafer上的电路板）来反求印刷这张图像的模板（Musk），其实这个光刻就像拍照片，拍的物体（比如风景）就是Musk,wafer就相当于胶卷，wafer上的图像就相当于风景在胶卷上的投缘。

第一个方向是，直接通过光学图像（也就是将要刻蚀在wafer上的电路板）来反求印刷这张图像的模板（Musk），我们的模型就是卷积网络，因为多层网络可以用来近似任何一个函数（通用近似定理 (Universal approximation theorem，一译万能逼近定理)」指的是：如果一个前馈神经网络具有线性输出层和至少一层隐藏层，只要给予网络足够数量的神经元，便可以实现以足够高精度来逼近任意一个在 Rn 的紧子集 (Compact subset) 上的连续函数。只要神经元足够多，再加一个激活函数），当然可以用来近似这个求逆过程。但是这条路风险很大，因此，我们有了第二条研究的路径，就是修正原有的物理模型，使得它更powerful，也就是说，我们任然需要建立物理model，来进行求逆过程，得到一个物理的model，只是，这个model不是足够强大，会有一些缺陷，因此，我们想通过一个CNN来修正这个model,来让我们的模型更有效。这就是大致的背景。背景讲清楚了，我们就该讲我们是怎么做的了。

搭建GPU环境，装显卡驱动（显卡驱动，蓝屏），cuda，cudnn，但是发现不能使用apt-get install东西，重装了很多次系统，装了不同的ubuntu系统，从12.04到16.04的五个不同版本，折腾了一周。（因此因为公司的IP保护，无线网卡解决问题）我们一开始做了些尝试，用了CNN，VGG，遇到的问题（图像预处理，亚像素的偏离），数据集不够，因为真实数据都是机密（IP），不可以轻易拿到。自己去切割图像，通过有经验的人工来做分类。再通过翻转这些操作来增加数据集。此外，数据集也是很讲究的，你得保证同一数据集中的物理参数，焦距，偏振方向都是一致的，焦距不同，得到的图像模糊程度不同。很长一段时间，我们都在用用记忆卷积，反卷积的深度神经网络，都市没有发现很好的效果。后来使用了pre-trained model,因为我们的数据量少（30000张图），但是经过CNN处理过的图像，并不比没有处理额图像好，这就尴尬了，因此，我们一直加深网络。

Linear Search Methods

the steepest descent algorithm proceeds as follows: at each step, starting from the point $x^{(k)}$,we conduct a line search in the direction $-\nabla f(x^{(k)})$, until a minimizer, $x^{(k+1)}$, is found.

$\alpha_k = argmin_{\alpha \geq 0} f(x_{(k)} - \alpha \nabla f(x^{(k)}))$

Proposition 8.1: if $x^{(k)}$ is the steepest descent sequence for f: R^n->R, then for each k the vector $x^{(k+1)} - x^{(k)}$ is orthogonal to the vector $x^{(k+2)} - x^{(k+1)}$

Proposition 8.2:if $x^{(k)}$ is the steepest descent sequence for f: R^n->R and if $\nabla f(x^{(k)}) \neq 0$, then

- $f(x^{(k+1)}) < f(x^{(k)})$

Stopping criterion:

- $\frac{|f(x^{(k+1)}) - f(x^{(k)})|}{||f(x^{(k)})||} < \epsilon$

Example: Quadratic function of the form:

- $f(x) = \frac{1}{2} x^T \mathrm{Q} x - b^T x$

Gradient:$g^{(k)} = \nabla f(x^{(k)}) = Qx - b$

so
- $x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$

where:

- $\alpha_k = argmin_{\alpha \geq 0} f(x_{(k)} - \alpha g^{(k)}) = argmin_{\alpha \geq 0} (\frac{1}{2} (x^k - \alpha g^{(k)})^T Q (x^k - \alpha g^{(k)}) - (x^k - \alpha$

Hence:

- $\alpha_k = \frac{g^{(k)T} g^{(k)}}{g^{(k)T} Q g^{(k)}}$

Covergence properties:
Define:

- $V(x) = f(x) + \frac{1}{2} x^{*T} Q x^* = \frac{1}{2} (x - x^*)^T Q (x - x^*)$

With:$x^* = Q^{-1} b$
Lemma 8.1 The iterative algorithm

- $x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$

with $g^{(k)} = Q x^{(k)} - b$ satisfies

- $V(x^{(k+1)}) = (1 - \gamma_k) V(x^{(k)})$,

where, if $g^{(k)} = 0$ then $\gamma_k = 1$, and if $g^{(k)} \neq 0$ then:

- $\gamma_k = \alpha_k \frac{g^{(k)T} Q g^{(k)}}{g^{(k)T} Q^{-1} g^{(k)}} (2 \frac{g^{(k)T} g^{(k)}}{g^{(k)T} Q g^{(k)}} - \alpha_k)$

Submit α_k into γ_k, then

- $\gamma_k = \frac{(g^{(k)T} g^{(k)})^2}{(g^{(k)T} Q g^{(k)})(g^{(k)T} Q^{-1} g^{(k)})}$

Theorem 8.1 Let $x^{(k)}$ be the sequence resulting from a gradient algorithm

$x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$. Let γ_k be as defined in Lemma 8.1, and suppose that $\gamma_k > 0$ for all k,

then $x^{(k)}$ converges to x^* for any initial condition $x^{(0)}$ if and only if:

- $\sum_{x=0}^{\infty} \gamma_k = \infty$

Theorem 8.2 In the steepest descent algorithm, we have

- $x^{(k)}$ ->x^* for any $x^{(0)}$

Theorem 8.3 For the fixed step size gradient algorithm, $x^{(k)}$ ->x^* for any $x^{(0)}$

if and only if $0 < \alpha < \frac{2}{\lambda_{max}(Q)}$

Convergence Rate:
Theorem 8.4 In the method of steepest descent applied to the quadratic function, at every step k, we have :

- $V(x^{(k+1)}) \leq (\frac{\lambda_{max}(Q) - \lambda_{min}(Q)}{\lambda_{max}(Q)}) V(x^{(k)})$.

Let:

- $r = \frac{\lambda_{max}(Q)}{\lambda_{min}(Q)} = ||Q||||Q^{-1}||$ the so-called condition number of Q.

Then, it follows from Theorem 8.4 that $V(x^{(k+1)}) \leq (1 - \frac{1}{r}) V(x^{(k)})$. We refer to $1 - \frac{1}{r}$ as the convergence ratio. If r = 1, then $\lambda_{max}(Q) = \lambda_{min}(Q)$, corresonding to circular contours of f. You can using the convergence ratio r to judge the speed of convergence.

Definition 8.1 Given a sequence $x^{(k)}$ thst converges to x^*, that is, $lim_{k->\infty} ||x^{(k)} - x^*|| = 0$, we say that the order of convergence is o, where $p \in R$, if $0 < lim_{k->\infty} \frac{x^{(k+1)} - x^*}{||x^{(k)} - x^*||^p} < \infty$,

if for all $p > 0$, $lim_{k->\infty} \frac{x^{(k+1)} - x^*}{||x^{(k)} - x^*||^p} = 0$,

then we say that the order of convergence is ∞.

Lemma 8.3. In the steepest descent algorithm, if $g(k) \neq 0$ for all k, then $\gamma_k = 1$ if and only if

$g(k)$ is an eigenvector of Q.

Theorem 8.6 Let $x^{(k)}$ be the sequence resulting from a gradient algorithm applied to a function f. Then, the order of convergence of $x^{(k)}$ is 1 in the worst case, that is, ehere exist a function f and an initial $x^{(0)}$ such that the order of convergence of $x^{(k)}$ is equal 1.

Newtow's Method

- $f(x) = f(x^{(x)}) + (x - x^{(k)})^T g^{(k)} + \frac{1}{2}(x - x^{(k)})^T F(x^{(k)})(x - x^{(k)})$

Theorem 9.1 Suppose that $f \in C^3$, and $x^* \in R^n$ is a point such that $\nabla f(x^*) = 0$ and $F(x^*)$ is invertible. Then, for all $x^{(0)}$ sufficiently close to x^*, Netwon's method is weel defined for all k, and converges to x^* with order of convergence at least 2.

At stated in the aboved theorem, Newton's methods has superies convergence properties if the starting point is near the solution. However, the method is not guaranteed to converge to the soulution if we start away from it(in fact, it may not even be well defined because the Hessian may be singular). In particular, the method may not be a descent method; that is, it is possible that $f(x^{(k+1)}) > f(x^{(k)})$. Fortunately, it is possible to modify the algorithm such that descent property holds. To see this, we need the following result.

Theorem 9.2 Let $x^{(k)}$ be the sequence generated by Newton's method for minimizing a given objective function f(x). If the Hessian $F(x^{(k)}) > 0$ and $g^{(k)} = \nabla f(x^{(k)}) \neq 0$, then the direction $d^{(k)} = -F(x^{(k)})^{-1} g^{(k)} = x^{(k+1)} - x^{(k)}$ from $x^{(k)}$ to $x^{(k+1)}$ is a descent direction for f in the sense that there exists an $\alpha > 0$ such that for all $a \in (a, \alpha)$,

$f(x^{(k)} = \alpha d^{(k)}) < f(x^{(k)})$.

The above theorem motivates the following modification of Newton's method:

- $x^{(k+1)} = x^{(k)} - \alpha_k F(x^{(k)})^{-1} g^{(k)}$ where
- $\alpha_k = argmin_{\alpha > 0} f(x^{(k) - \alpha_k F(x^{(k)})^{-1} g^{(k)}})$

A drawback of Newton's method is that evaluation of $F(x^{(k)})$ for large n can be computationally expensive. Furthermore, we have to solve the set of n linear equations $F(x^{(k)}) d^{(k)} = -g^{(k)}$. In the Chapters 10 and 11, we discuss methods that alleviate this difficulty.

Levenberg-Marquardt Modification:

If the Hessian matrix $F(x^{(k)})$ is not positive definite, then the search direction

$d^{(k)} = -F(x^{(k)})^{-1}g^{(k)}$ may not point in a descent direction. A simple technique to ensure that the search direction is a descent direction is to introduce the so-called Levenberg-Marquardt Modification to Newton's algorithm:

$x^{(k+1)} = x^{(k)} - \alpha_k F(x^{(k)} + u_k I)^{-1} g^{(k)}$

where $u_k \geq 0$

Newton's methods for nonlinear least-squares

Consider the following problem:

- $minimize \sum_{i=1}^{m} (r_i(x))^2$

where $r_i : R^n -> R, i = 1, ..., m$, are given functions. This particular problem is called a nonlinear least-squares problem

- $F_{jk} = 2 \sum_i (\frac{\partial r_i}{\partial x_j} \frac{\partial r_i}{\partial x_k} + r_i \frac{\partial^2 r_i}{\partial x_j \partial x_k})$,

Let $s(x) = r_i(x) \frac{\partial^2 r_i}{\partial x_j \partial x_k}(x)$
So the Hessian matrix as

- $F(x) = 2(J(x)^T J(x) + S(x))$.

Therefore, Newton's method applied to the nonlinear least-squares problems is given by

- $x^{(k+1)} = x^{(k)} - (J(x)^T J(x) + S(x))^{-1} J(x)^T r(x)$. In some case, s(x) can ne ignored because its components are negligibly small. In this case, the above Newtons's algorithm reduces to what is commonly called the Gauss-Newtons method:

$x^{(k+1)} = x^{(k)} - (J(x)^T J(x))^{-1} J(x)^T r(x)$.

A potential problem with the Gauss-Newton method is that the matri $J(x)^T J(x)$ may not be positive definite. As described before, this problem can be overcome using a Levenberg-Marquardt modification:

- $x^{(k+1)} = x^{(k)} - (J(x)^T J(x) + u_k I)^{-1} J(x)^T r(x)$.

Conjugate Direction Methods

The conjugate direction methods typically perrform better than the method of steepest descent, but not as weel as Newton's method. As we saw from the method of steepest descent and Newton's method, the crucial factor in the efficiency of an iterative search method is the direction of the search at each iteration.

Definition 10.1 Let Q be a real symmetric nxn matrix. The directions $d^{(0)}, d^{(1)}, d^{(2)}, ..., d^{(m)}$ are Q-conjugate if, for all $i \neq j$, we have $d(i)TQd^{(j)} = 0$.

Lemma 10.1 Let Q be as symmetric positive definite nxn matrix. If the diretions $d^{(0)}, d^{(1)}, d^{(2)}, ..., d^{(m)} \in R^n, k \leq n-1$, are nonzero and Q-conjugate, then they are linearly independent.

Basic conjugate Direction Algorithm. Given a start $x^{(0)}$, and Q-conjugate direction

- $d^{(0)}, d^{(1)}, d^{(2)}, ..., d^{(m)}, for k \geq 0,$
- $g^{(k)} = \nabla f(x^{(k)}) = Qx^{(k)} - b,$
- $\alpha_k = -\frac{g^{(k)T} d^{(k)}}{d^{(k)T} Q d^{(k)}}$
- $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$

Theorem 10.1 For any starting point $x^{(0)}$, the basic conjugate direction algorithm converges to the unique x^* (that solves Qx = b) in n steps; that is, $x^{(n)} = x^*$

Lemma 10.2 In the conjugate direction algorithm, $g^{(k+1)T} d^{(i)} = 0$ for all k, $0 \leq k \leq n-1$, and $0 \leq i \leq k$

The conjugate gradient algorithm is summarized below.

1. Set k := 0; select the initial point $x^{(0)}$
2. $g^{(0)} = \nabla f(x^{(0)})$. If $g^{(0)} = 0$, stop, else set $d^{(0)} = -g^{(0)}$. 3. $\alpha_k = -\frac{g^{(k)T} d^{(k)}}{d^{(k)T} Q d^{(k)}}$
3. $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$
4. $g^{(k+1)} = \nabla f(x^{k+1})$. If $g^{(k+1)} = 0$,stop.
5. $\beta_k = -\frac{g^{(k+1)T} Q d^{(k)}}{d^{(k)T} Q d^{(k)}}$ 7. $d^{(k+1)} = -g^{(k+1)} + \beta_k d^{(k)}$
6. Set k:= k+1; go to step 3.

Proposition 10.1 In the conjugate gradient algorithm, the directions $d^{(0)}, d^{(1)}, d^{(2)}, ..., d^{(n-1)}$ are Q-conjugate.

LTR

LTR or machine-learned ranking(MLR)是运用机器学习，典型的监督，半监督或者强化学习来为信息检索系统构建排序模型

排序学习可以在信息检索（IR）,NLP, DM等领域被广泛应用，典型的应用有文献检索，专家检索系统，定义查询系统，协同过滤，问答系统，关键词提取，文档摘要还有机器翻译等。

互联网搜索方面的一个新趋势是使用机器学习方法去自动的建立评价模型f(q, d)。

对于标注训练集，选定LTR方法，确定损失函数，以最小化损失函数为目标进行优化即可得到排序模型的相关参数，这就是学习过程。预测过程就是将待预测结果输入到学习到的排序模型中，即可以得到结果的相关得，利用该得分进行排序即可得到待预测结果的最终顺序。

最优化

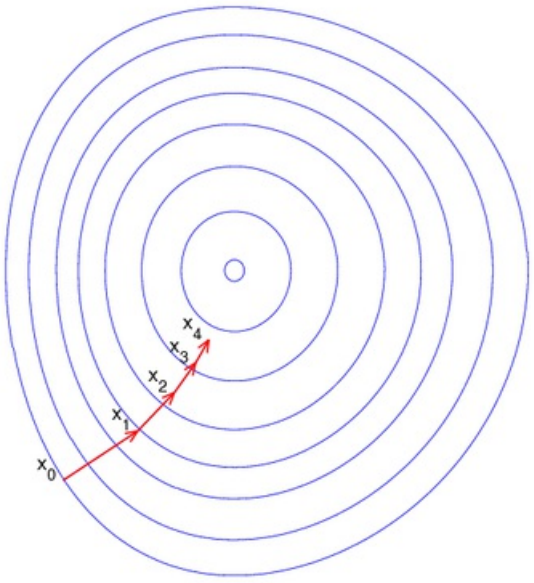
最速下降法，牛顿法，LBFGS

1.梯度下降法（Gradient Descent）

梯度下降法是最早最简单，也是最为常用的最优化方法。梯度下降法实现简单，当目标函数是凸函数时，梯度下降法的解是全局解。一般情况下，其解不保证是全局最优解，梯度下降法的速度也未必是最快的。

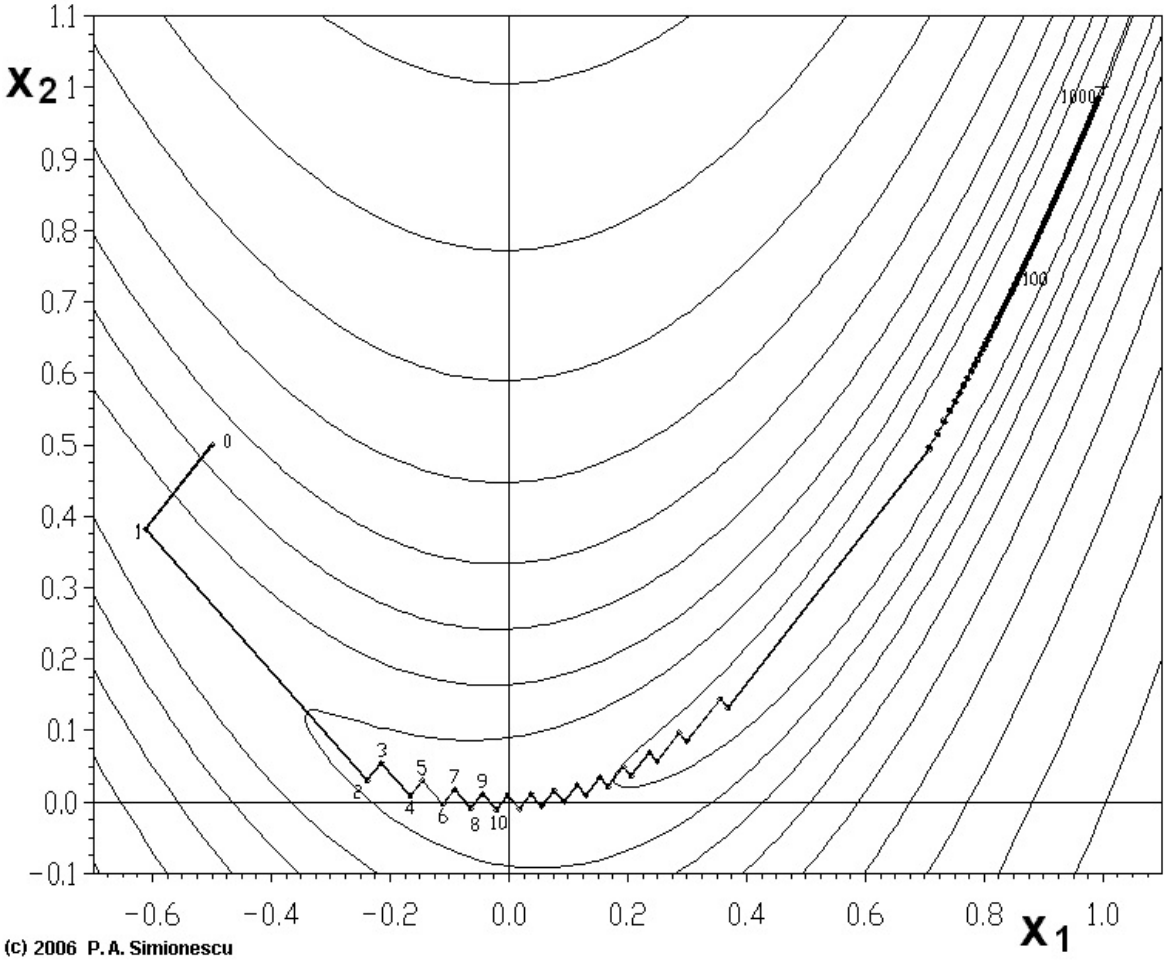
梯度下降法的优化思想是用当前位置负梯度方向作为搜索方向，因为该方向为当前位置的最快下降方向，所以也被称为是"最速下降法"。最速下降法越接近目标值，梯度趋于0，所以步长越小，前进越慢。

梯度下降法的搜索迭代示意图如下图所示：



梯度下降法的缺点：

（1）越靠近极小值的地方收敛速度越慢，如下图所示



从上图可以看出，梯度下降法在接近最优解的区域收敛速度明显变慢，利用梯度下降法求解需要很多次的迭代。

在机器学习中，基于基本的梯度下降法发展了两种梯度下降方法，分别为随机梯度下降法和批量梯度下降法。

比如对一个线性回归（Linear Logistics）模型，假设下面的h(x)是要拟合的函数，J(theta)为损失函数，theta是参数，要迭代求解的值，theta求解出来了那最终要拟合的函数h(theta)就出来了。其中m是训练集的样本个数，n是特征的个数。

$$h(\theta)=\sum_{j=0}^{n}\theta_k x_j \qquad J(\theta)=\frac{1}{2m}\sum_{i=0}^{m}(y^i-h_\theta(x^i))^2$$

//n是特征的个数，m的训练样本的数目

1)批量梯度下降法（Batch Gradient Descent, BDG）

(1)将$J(\theta)$对θ求偏导，得到每个θ对应的梯度：

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

(2)由于要最小化风险函数，所以按照每个参数θ的负梯度方向来更新θ

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

(3)从上面公式可以注意到，它得到的是一个全局最优解，但是每迭代一步，都要用到训练集所有的数据，如果m很大，那么可想而知这种方法的迭代速度会相当的慢。所以，这就引入了另外一种方法——随机梯度下降。
一个实验结果，说明随机梯度下降法能收敛到最终结果：

对于批量梯度下降法，样本个数m，x为n维向量，一次迭代需要把m个样本全部带入计算，迭代一次计算量为m*n2。

2)随机梯度下降（Stochastic Gradient Descent, SGD）

(1)上面的风险函数可以写成如下这种形式，损失函数对应的是训练集中每个样本的粒度，而上面批量梯度下降对应的是所有的训练样本

- $J(\theta) = \frac{1}{2m}\sum_{i=0}^{m}(y^i - h_\theta(x^i))^2 = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^i, y^i))$
- $cost(\theta, (x^i, y^i)) = \frac{1}{2}(y^i - h_\theta(x^i))^2$

(2)每个样本的损失函数，对θ求偏导得到对应的梯度，来更新θ

- $\theta_j' = \theta_j + (y^i - h_\theta(x^i))x_j^i$

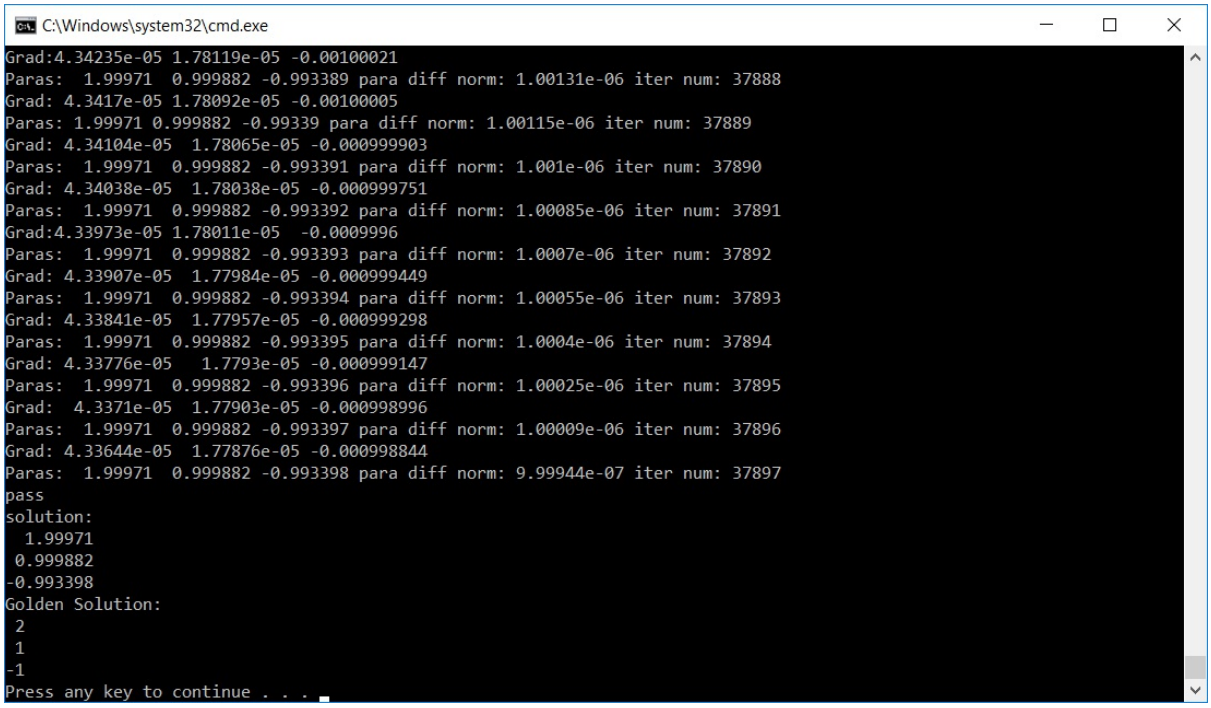
(3)随机梯度下降是通过每个样本来迭代更新一次，如果样本量很大的情况（例如几十万），那么可能只用其中几万条或者几千条的样本，就已经将θ迭代到最优解了，对比上面的批量梯度下降，迭代一次需要用到十几万训练样本，一次迭代不可能最优，如果迭代10次的话就需要遍历训练样本10次。但是，SGD伴随的一个问题是噪音较BGD要多，使得SGD并不是每次迭代都向着整体最优化方向。
随机梯度下降每次迭代只使用一个样本，迭代一次计算量为n2，当样本个数m很大的时候，随机梯度下降迭代一次的速度要远高于批量梯度下降方法。两者的关系可以这样理解：随机梯度下降方法以损失很小的一部分精确度和增加一定数量的迭代次数为代价，换取了总体的优化效率的提升。增加的迭代次数远远小于样本的数量。

比较批量梯度下降法与随机梯度下降法：

用$y = 2 * x1 + x2 - 1 + rand(0,1)$产生1000组数，用这一组数据来反求产生函数中的系数$(2,1,-1)^T$.

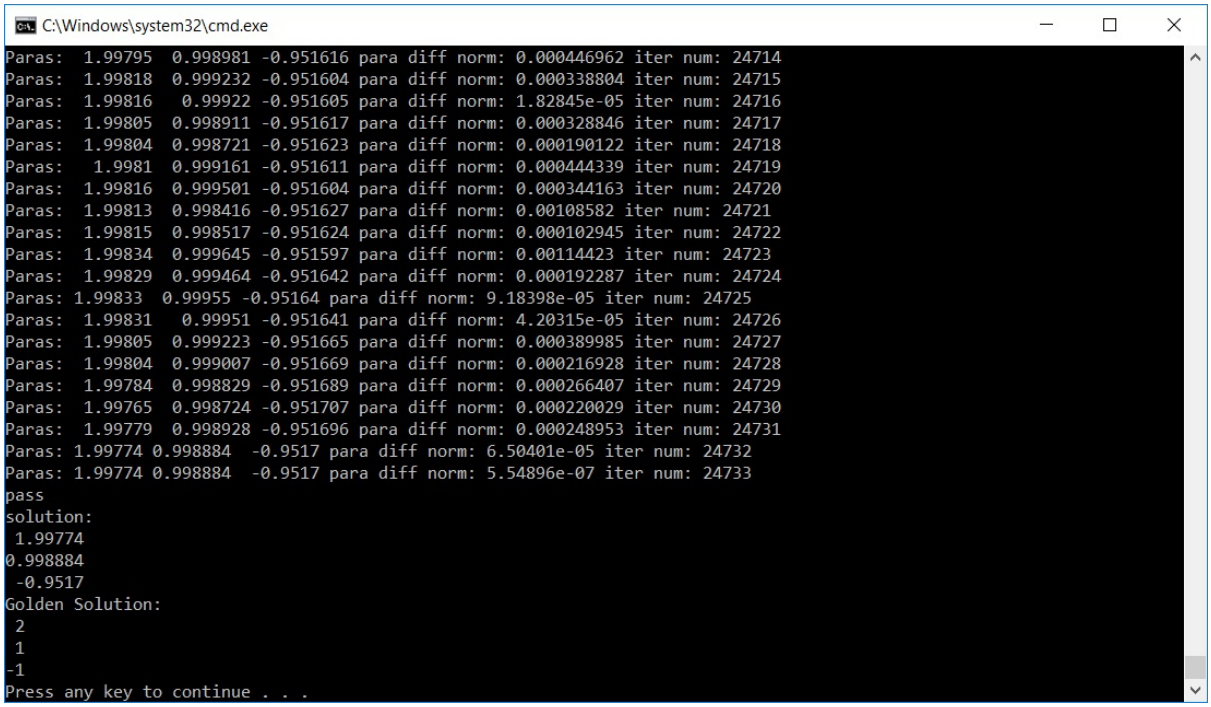
迭代停止条件就是，训练得到的相邻两次参数的差的范数小于0.000001(严格来说，停止条件是损失函数一介导数等于0，也就是$L'(\theta) = -\frac{1}{m}X^T(X\theta - Y) = 0$).

1)批量梯度下降法：1000个样本，设置的learn rate = 0.0001，一次迭代所有1000个样本，最终经过38000次迭代收敛到可以接受的标准。如果设置learn rate 是较大的值，比如为1，则发现结果马上发散了，因此批量梯度下降法对learn rate比较敏感，而下面的随机梯度下降法就不会出现这个问题，因为每次只对一个样本进行处理，即使这个样本对参数的梯度很大，但是下一个样本又可以马上把参数拉回来。

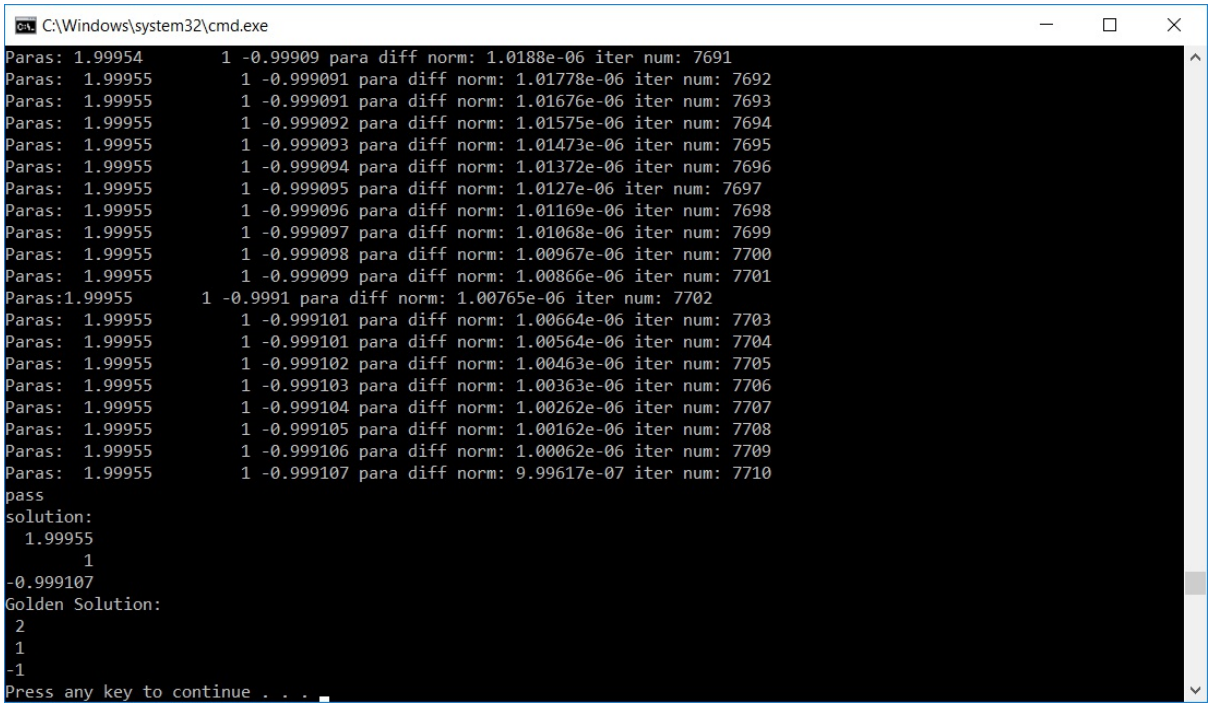


2)随机梯度下降法：

1000个样本，设置的learn rate = 1，一次迭代处理一个样本，最终经过25000次迭代收敛到可以接受的标准。



3)牛顿法



对批量梯度下降法和随机梯度下降法的总结：

批量梯度下降---最小化所有训练样本的损失函数，使得最终求解的是全局的最优解，即求解的参数是使得风险函数最小，但是对于大规模样本问题效率低下。

随机梯度下降---最小化每条样本的损失函数，虽然不是每次迭代得到的损失函数都向着全局最优方向，但是大的整体的方向是向全局最优解的，最终的结果往往是在全局最优解附近，适用于大规模训练样本情况。

从实验数据来看，批量梯度下降法收敛到的是全局最优值，而随机梯度下降法收敛到最优值附件的地方。随机梯度下降法的好处是收敛远快于批量梯度下降法。

问题可以转化成寻找一组λ使得:$L(\lambda) = (X * \lambda - Y)^T (X * \lambda - Y)$极小

利用$L(\lambda)$对向量λ求导可以得到[1]:

- $\frac{\partial L(\lambda)}{\partial \lambda} = 2X^T(X\lambda - Y) = 0$

我们也可以得到:

- $\frac{\partial^2 L(\lambda)}{\partial \lambda^2} = 2X^T X$
- X(n+1) = X(n) - f'(X(n))/f''(X(n))

```
MatrixXd Iteration::BathGradDescent(MatrixXd paras, int iterator_num)
{
    int rows = m_input_data.rows();
    int cols = m_input_data.cols();
    MatrixXd init_paras = paras;
    if (m_input_data.cols() == paras.rows())
    {

        MatrixXd grad = (1.0 / rows)*m_input_data.transpose()*(m_output_data - m_input_data*i
nit_paras);
        paras += m_learn_rate*grad;
    }
    else
    {
        //error
    }
    return paras;
}

MatrixXd Iteration::StocGradDescent(MatrixXd paras, int index)
{
    int rows = m_input_data.rows();
    int cols = m_input_data.cols();
    MatrixXd input_data_i(1,cols);
    MatrixXd output_data_i(1, 1);
    output_data_i(0, 0) = m_output_data(index,0);

    for (int i = 0; i < cols; i++)
    {
        input_data_i(0, i) = m_input_data(index, i);
    }


    if (m_input_data.cols() == paras.rows())
    {
        MatrixXd grad = (1.0 / rows) *input_data_i.transpose()*(input_data_i*paras - output_d
ata_i);
        paras = paras - grad;
    }
    else
    {
```

```
            //error
        }
        return paras;
    }
```

由于牛顿法是基于当前位置的切线来确定下一次的位置，所以牛顿法又被很形象地称为是"切线法"。牛顿法的搜索路径（二维情况）如下图所示：

牛顿法搜索动态示例图：

牛顿法和拟牛顿法（Newton's method &Quasi-Newton methods）

1)牛顿法（Newton's method）
牛顿法是一种在实数域和复数域上近似求解方程的方法。方法使用函数f (x)的泰勒级数的前面几项来寻找方程f (x) = 0的根。牛顿法最大的特点就在于它的收敛速度很快。[1]
步骤：

首先，选择一个接近函数f(x)零点的x_0,计算相应的计算相应的 $f(x_0)$ 和切线斜率$f'(x_0)$。然后计算穿过点$(x_0, f(x_0))$并且斜率为$f'(x_0)$的直线和x轴的交点的x轴坐标，也就是如下方程：

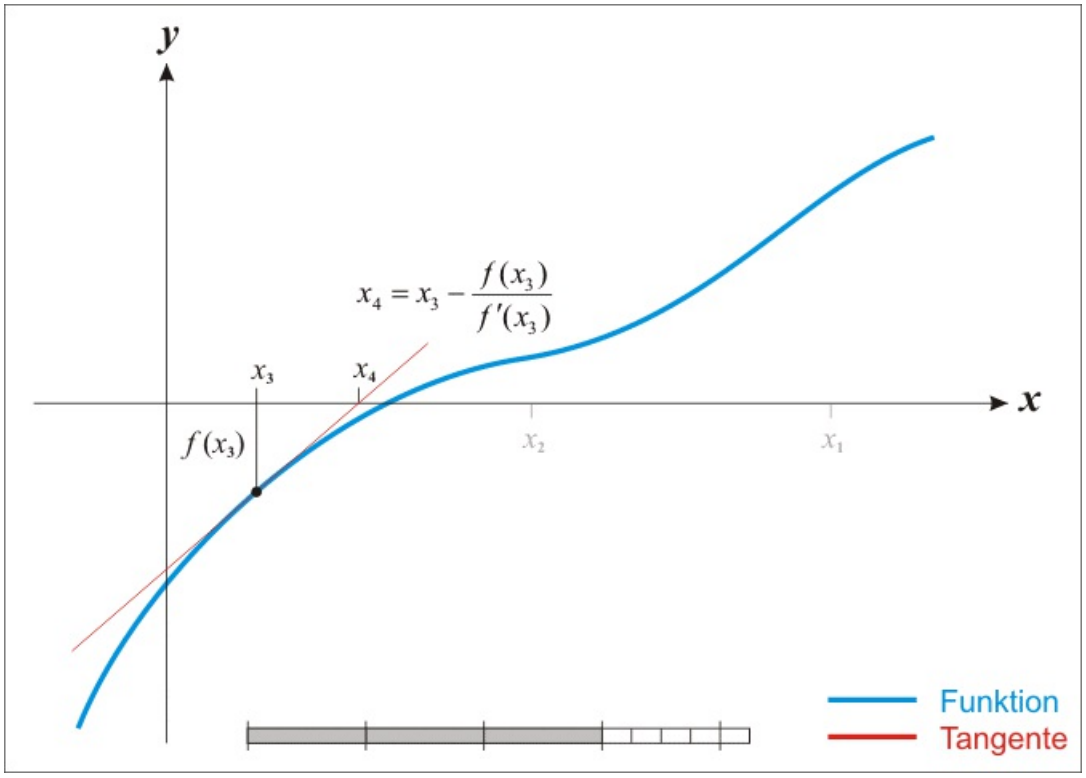
- $x * f'(x_0) + f(x_0) - x_0 * f'(x_0) = 0$

我们将新求得的点的 x 坐标命名为x_1，通常x_1会比x_0更接近方程$f(x) = 0$的解。因此我们现在可以利用x_1开始下一轮迭代。迭代公式可化简为如下所示：

- $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
- $\mathbf{X_{n+1}} = \mathbf{X_n} - \frac{1}{1}$

如果X是向量，则可以写成向量形式：

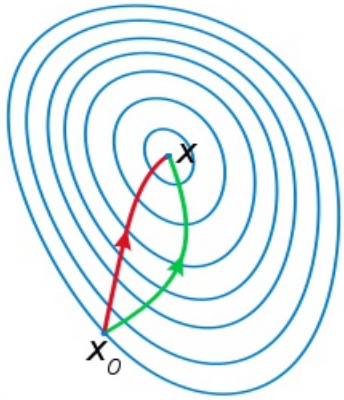
已经证明，如果$f'(x)$是连续的，并且待求的零点x是孤立的，那么在零点x周围存在一个区域，只要初始值x_0位于这个邻近区域内，那么牛顿法必定收敛。并且，如果$f'(x)$不为0，那么牛顿法将具有平方收敛的性能. 粗略的说，这意味着每迭代一次，牛顿法结果的有效数字将增加一倍。下图为一个牛顿法执行过程的例子。



关于牛顿法和梯度下降法的效率对比：

从本质上去看，牛顿法是二阶收敛，梯度下降是一阶收敛，所以牛顿法就更快。如果更通俗地说的话，比如你想找一条最短的路径走到一个盆地的最底部，梯度下降法每次只从你当前所处位置选一个坡度最大的方向走一步，牛顿法在选择方向时，不仅会考虑坡度是否够大，还会考虑你走了一步之后，坡度是否会变得更大。所以，可以说牛顿法比梯度下降法看得更远一点，能更快地走到最底部。（牛顿法目光更加长远，所以少走弯路；相对而言，梯度下降法只考虑了局部的最优，没有全局思想。）

根据wiki上的解释，从几何上说，牛顿法就是用一个二次曲面去拟合你当前所处位置的局部曲面，而梯度下降法是用一个平面去拟合当前的局部曲面，通常情况下，二次曲面的拟合会比平面更好，所以牛顿法选择的下降路径会更符合真实的最优下降路径。



注：红色的牛顿法的迭代路径，绿色的是梯度下降法的迭代路径。

牛顿法的优缺点总结：

优点：二阶收敛，收敛速度快；

缺点：牛顿法是一种迭代算法，每一步都需要求解目标函数的**Hessian**矩阵的逆矩阵，计算比较复杂。

2）拟牛顿法（Quasi-Newton Methods）

拟牛顿法是求解非线性优化问题最有效的方法之一，于20世纪50年代由美国Argonne国家实验室的物理学家W.C.Davidon所提出来。Davidon设计的这种算法在当时看来是非线性优化领域最具创造性的发明之一。不久R. Fletcher和M. J. D. Powell证实了这种新的算法远比其他方法快速和可靠，使得非线性优化这门学科在一夜之间突飞猛进。

拟牛顿法的本质思想是改善牛顿法每次需要求解复杂的Hessian矩阵的逆矩阵的缺陷，它使用正定矩阵来近似Hessian矩阵的逆，从而简化了运算的复杂度。拟牛顿法和最速下降法一样只要求每一步迭代时知道目标函数的梯度。通过测量梯度的变化，构造一个目标函数的模型使之足以产生超线性收敛性。这类方法大大优于最速下降法，尤其对于困难的问题。另外，因为拟牛顿法不需要二阶导数的信息，所以有时比牛顿法更为有效。如今，优化软件中包含了大量的拟牛顿算法用来解决无约束，约束，和大规模的优化问题。

具体步骤：

拟牛顿法的基本思想如下。首先构造目标函数在当前迭代x_k的二次模型：

- $m_k(p) = f(x_k) + \bigtriangledown f(x_k)^T p + \frac{p^T B_k p}{2}$
- $p_k = -B_k^{-1} \bigtriangledown f(x_k)$

这里B_k是一个对称正定矩阵，于是我们取这个二次模型的最优解作为搜索方向，并且得到新的迭代点：

- $x_{k+1} = x_k + \alpha_k p_k$

其中我们要求步长α_k满足Wolfe条件。这样的迭代与牛顿法类似，区别就在于用近似的Hesse矩阵B_k 代替真实的Hesse矩阵。所以拟牛顿法最关键的地方就是每一步迭代中矩阵Bk 的更新。现在假设得到一个新的迭代$x_k + 1$，并得到一个新的二次模型：

这个公式被称为割线方程。常用的拟牛顿法有DFP算法和BFGS算法。

最优化方法：牛顿迭代法和拟牛顿迭代法

共轭梯度法

共轭梯度法（Conjecture Gradient）是介于最速下降法与牛顿法之间的一个方法，它仅需要一阶导数信息，但克服了最速下降法收敛慢的缺点，同时又避免了牛顿法需要储存和计算Hesse矩阵并求逆的缺点，共轭梯度法不仅是解决大型线性方程组最有用的方法之一，也是解大型非线性最优

化问题最有效的算法之一。向量共轭的定义：若$\mathbf{A}$是正定对称阵，若非0矢量$\mathbf{u}, \mathbf{v}$满足，$\mathbf{u^T A v} = 0$，则称矢量$\mathbf{u}, \mathbf{v}$是共轭的。假设：$P = [\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_n}]$是一组基于$\mathbf{A}$的共轭矢量，则$\mathbf{Ax} = \mathbf{b}$的解$\mathbf{x}^*$可以表示为：$\mathbf{x}^* = \sum_{i=1}^{n} \alpha_i \mathbf{P_i}$,因此基于基矢量展开，我们有：

$\mathbf{Ax}_* = \sum_{i=1}^{n} \alpha_i \mathbf{AP_i}$ 左乘$\mathbf{P_k^T}$：$\mathbf{P_k^T Ax}_* = \sum_{i=1}^{n} \alpha_i P_k^T \mathbf{AP_i}$ 代入：

$\mathbf{P_k^T Ax}_* = \mathbf{b}$,$\mathbf{u^T Av} = \langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{A}}$,利用$i! = k$有$\langle \mathbf{p_k}, \mathbf{p_i} \rangle_{\mathbf{A}} = \mathbf{0}$ 就得到：

如果我们的小心的选择$\mathbf{p_k}$,为了获得解$\mathbf{x}_*$的一个好的近似，我们并不需要所有的基向量。因此，我们把共轭梯度算法当成一种迭代算法，它也允许我们近似n很大，以至于直接求解需要花费太多时间的系统。我们给$\mathbf{x}_*$一个初始猜测值$\mathbf{x_0}$，假设$\mathbf{x_0} = \mathbf{0}$,在求解的过程中，我们需要一种标准来告诉我们是否我们的解更靠近真实的解$\mathbf{x}_*$,实际上，求解$\mathbf{Ax} = \mathbf{b}$等价于求解如下二次函数的极小值：$\mathbf{f(x)} = \frac{\mathbf{1}}{\mathbf{2}} \mathbf{x^T Ax} - \mathbf{x^T b}$，他存在唯一的最小值，因为他的二阶导是正定的：

$\mathbf{D^2 f(x)} = \mathbf{A}$，解就是一阶导数等于0的地方 $\mathbf{Df(x)} = \mathbf{Ax} - \mathbf{b}$，假设第一个基矢$\mathbf{p_0}$就是$f(x)$在$\mathbf{x} = \mathbf{x_0}$处的负梯度，我们可以假设

$\mathbf{x_0} = \mathbf{0}$.因此：$\mathbf{p_0} = \mathbf{b} - \mathbf{Ax_0}$. 令$\mathbf{r_0}$表示第k步的残差：

$\mathbf{r_k} = \mathbf{b} - \mathbf{Ax_k}$,$\mathbf{r_k}$是$\mathbf{f(x)}$在$\mathbf{x} = \mathbf{x_k}$处的负梯度。为了让$\mathbf{p_k}$与前面的所有的$\mathbf{p_i}$相互共轭，$\mathbf{p_k}$可以如下构造：$\mathbf{p_k} = \mathbf{r_k} - \sum_{\mathbf{i<k}} \frac{\mathbf{p_i^T A r_k}}{\mathbf{p_i^T A p_i}}$ 沿着这个方向，因此下一步的优化方向就是：

$\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_\mathbf{k} \mathbf{p_k}$,满足：$g'(\alpha_k = 0)$ 因此：

$\mathbf{f(x_{k+1})} = \mathbf{f(x_k} + \alpha_\mathbf{k} \mathbf{p_k)} =: \mathbf{g}(\alpha_\mathbf{k})$，$\alpha_\mathbf{k} = \frac{\mathbf{p_k^T b}}{\mathbf{p_k^T A p_k}} = \frac{\mathbf{p_k^T (r_k + Ax_k)}}{\mathbf{p_k^T A p_k}} = \frac{\mathbf{p_k^T r_k}}{\mathbf{p_k^T A p_k}}$ 算法：

$\mathbf{r_0} := \mathbf{b} - \mathbf{Ax_0}$ $\mathbf{p_0} := \mathbf{r_0}$ $\mathbf{k} := \mathbf{0}$ repeat: $\alpha_\mathbf{k} := \mathbf{r_0}$

$\alpha_\mathbf{k} := \frac{\mathbf{r_k^T r_k}}{\mathbf{p_k^T A p_k}}$ $\mathbf{x_{k+1}} := \mathbf{x_k} + \alpha_\mathbf{k} \mathbf{p_k}$ $\mathbf{r_{k+1}} := \mathbf{r_k} + \alpha_\mathbf{k} \mathbf{A p_k}$ if r_{k+1} is sufficiently small, then exit loop. $\beta_\mathbf{k} := \frac{\mathbf{r_{k+1}^T r_{k+1}}}{\mathbf{r_k^T r_k}}$ $\mathbf{p_{k+1}} := \mathbf{r_{k+1}} + \beta_\mathbf{k} \mathbf{p_k}$

$\mathbf{k} := \mathbf{k} + \mathbf{1}$

protected Vector FillDataAbsolute(Vector f, Matrix J, Variable variable)

线性与非线性方程组解的稳定性分析

线性方程组解的稳定性分析

系统解的稳定性定义成系统小的扰动对系统解的影响，。

Example 1:

$$\begin{bmatrix} 400 & -201 \\ -800 & 401 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 200 \\ -200 \end{bmatrix}$$

Solution:

$x_1 = -100, x_2 = -200$

If we give small change to matrix A, change $\mathbf{A_{11}}$from 400 to 401 then:

$$\begin{bmatrix} 401 & -201 \\ -800 & 401 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 200 \\ -200 \end{bmatrix}$$

This time the solution is:

$x_1 = 40000, x_2 = 79800$

Ill-conditiond:

When the solution is highly sensitive to the values of the coefficient matrix A or the righthand side constant vector b, the equations are called to be ill-conditioned.

Condition Number

Let's linear equations:

- $\mathbf{Ax} = \mathbf{b}$,

Let us investigate first, how a small change in the $\mathbf{b}$ vector changes the solution vector. $\mathbf{x}$ is the solution of the original system and let $\mathbf{x} + \mathbf{\Delta x}$ is the solution when $\mathbf{b}$ changes from $\mathbf{b}$ to $\mathbf{b} + \mathbf{\Delta b}$.

Then we can write:

- $\mathbf{A}(\mathbf{x} + \mathbf{\Delta x}) = \mathbf{b} + \mathbf{\Delta b}$

or

- $\mathbf{Ax} + \mathbf{A\Delta x} = \mathbf{b} + \mathbf{\Delta b}$

But because $\mathbf{Ax} = \mathbf{b}$, it follows that

- $\mathbf{A\Delta x} = \mathbf{\Delta b}$

So:

- $\delta \mathbf{x} = \mathbf{A^{-1}\Delta b}$

Using the matrix norm properties:

- $||\mathbf{Ax}|| \leq ||\mathbf{A}||||\mathbf{x}||$

We can get:

- $||\mathbf{A^{-1}\Delta b}|| \leq ||\mathbf{A^{-1}}||||\mathbf{\Delta b}||$,

Also, we can get:

- $||\mathbf{b}|| = ||\mathbf{Ax}|| \leq ||\mathbf{A}||||\mathbf{x}||$

Using equations 2.5 and 2.6 we can get:

- $\frac{||\mathbf{\Delta x}||}{||\mathbf{A}||\cdot||\mathbf{x}||} \leq \frac{||\mathbf{A^{-1}}||\cdot||\mathbf{\Delta b}||}{||\mathbf{b}||}$

Let's define condition number as: $\mathbf{K(A)} = ||\mathbf{A}|| \cdot ||\mathbf{A^{-1}}||$

we can rewrite equation 2.7 as:

- $\frac{||\mathbf{\Delta x}||}{||\mathbf{x}||} \leq \mathbf{K(A)} \cdot \frac{||\mathbf{\Delta b}||}{||\mathbf{b}||}$

Now, let us investigate what happens if a small change is made in the coefficient matrix

- $\mathbf{A}$. Consider $\mathbf{A}$ is changed to $\mathbf{A} + \mathbf{\Delta A}$ and the solution changes from $\mathbf{x}$ to $\mathbf{x} + \mathbf{\Delta x}$
- $(\mathbf{A} + \mathbf{\Delta A})(\mathbf{x} + \mathbf{\Delta x}) = \mathbf{b}$, we can obtain:

- $\frac{||\mathbf{\Delta x}||}{||\mathbf{x+\Delta x}||} \leq \mathbf{K(A)} \cdot \frac{||\mathbf{\Delta A}||}{||\mathbf{A}||}$

$\mathbf{K(A)}$ is a measure of the relative sensitivity of the solution to changes in the right-hand side vector b.When the condition number $\mathbf{K(A)}$ becomes large, the system is regarded as being illconditioned.
Matrices with condition numbers near 1 are said to be well-conditioned.

非线性方程组解的稳定性分析

For non-linear system,

最小二乘法求解线性，非线性方程组

所有的线性方程组或者非线性方程组可以转化成一个最小二乘法问题，使得拟合的方程与观察值之间的平方和最小。

- $\hat{\beta} = \mathbf{argmin}_{\beta}\mathbf{S}(\beta) = \mathbf{argmin}_{\beta} \sum_{\mathbf{i=1}}^{\mathbf{m}}[\mathbf{y_i} - \mathbf{f}(\mathbf{x_i}, \beta)]^{\mathbf{2}}$

y_i是观测值，x_i是已知变量，一共m组观测值。

当$\mathbf{f}(\mathbf{x}, \beta)$是线性方程组时

即 $\mathbf{f}(\mathbf{X},\beta)=\mathbf{A}\beta$
cost function 可以表示如下:

$$\mathbf{L}(\beta)=||\mathbf{Y}-\mathbf{A}\beta||^{\mathbf{2}}$$

其中$\mathbf{Y}=[\mathbf{y_1},\mathbf{y_2}\cdots\mathbf{y_m}]^{\mathbf{T}}$
cost function 对β

$$\mathbf{L}(\beta)=\mathbf{Y^TY}-\mathbf{2Y^TA}\beta+\beta^{\mathbf{T}}\mathbf{A^TA}\beta$$

求导,

$$-2A^TY+2(A^TA)\beta=0$$

再让导数等于0,就可以求得解为:β

$$\beta=(\mathbf{A^TA})^{-1}\mathbf{A^TY}$$

我们接下来主要讨论非线性的情况:

也就是残差$\mathbf{y_i}-\mathbf{f}(\mathbf{x_i},\beta)$不能表示成线性形式。
当然我们有很多方法来求解方程(4.1),比如 梯度下降法, 牛顿法, 高斯-牛顿法, 以及 Levenberg-Marquardt 梯度下降法
把损失函数展开到一阶。

$$f(x+\alpha\mathbf{d})=f(x_0)+\alpha f'(x_0)\mathbf{d}+O(\alpha^2)$$

$$x_{t+1}=x_t-\alpha f'(x_t),$$

牛顿法

把cost function进行展开到二阶:

$$f(x_{t+1})=f(x_t)+g(x_{t+1}-x_t)+\frac{1}{2}(x_{t+1}-x_t)^TH(x_{t+1}-x_t)$$

求导, $\frac{\partial f}{\partial x_t}=g+H(x_{t+1}-x_t)$,让导数为0就有

$$x_{t+1}=x_t-H^{-1}g$$

要是H是正定的, 上面的就是凸函数, 也就一定有了最小值。可惜H不一定是正定的, 这就引导出了下面的方法

高斯-牛顿法

cost function可以表示成残差的形式:

$$L(x)=\sum_{i=1}^{m}r_i(x)^2$$

根据牛顿法:

$vx_{t+1}=x_t-H^{-1}g,$
梯度表示为:

$$\partial r$$

- $g_j = 2\sum_i r_i \frac{\partial r_i}{\partial x_j}$

对方程(4.9)求二阶导，我们可以得到Hessian矩阵：

- $H_{jk} = 2\sum_i (\frac{\partial r_i}{\partial x_j}\frac{\partial r_i}{\partial x_k} + r_i \frac{\partial^2 r_i}{\partial x_j \partial x_k}),$

当残差r_i很小的时候,我们就可以去掉最后一项，因此

- $H_{jk} \approx 2\sum_i J_{ij}J_{ik} \quad With \quad J_{ij} = \frac{\partial r_i}{\partial x_j}$

这样Hessian就是半正定了。
因此参数迭代公式(4.10)可以写成：

- $x_{t+1} = x_t - (J^T J)^{-1} J^T r,$

Levenberg-Marquardt
方法用于求解非线性最小二乘问题，结合了梯度下降法和高斯-牛顿法。
其主要改变是在Hessian阵中加入对角线项。当然，这是一种L2正则化方式。

- $x_{t+1} = x_t - (H + \lambda I_n)^{-1} g,$
- $x_{t+1} = x_t - (J^T J + \lambda I_n)^{-1} J^T r$

L-M算法的不足点。

当λ很大时，$J^T J + \lambda I_n$根本没用，Marquardt为了让梯度小的方向移动的快一些，来防止小梯度方向的收敛，把中间的单位矩阵换成了$diag(J^T J)$,因此迭代变成：

- $x_{t+1} = x_t - (J^T J + \lambda diag(J^T J))^{-1} J^T r$

阻尼项(damping parameter)λ的选择，Marquardt 推荐一个初始值λ_0与因子$v > 1$，开始时，

$\lambda = \lambda_0$,然后计算cost functions,第二次计算$\lambda = \lambda_0 / v$,如果两者cost function都比初始点高，然后我们就增大阻尼项，通过乘以v,直到我们发现当$\lambda = \lambda_0 v^k$时，cost function下降。

如果使用$\lambda = \lambda_0 / v$使得cost fucntion下降，然后我们就把λ_0 / v作为新的λ
当$\lambda = 0$时，就是高斯-牛顿法，当λ趋于无穷时，就是梯度下降法。如果使用λ / v没有是损失函数下降，使用λ导致损失函数下降，那么我们就继续使用λ做为阻尼项。

归一化的残差

如果我们假设数据中每一点的贡献是等权重，因此，我们有必要对每一项残差加一个权重，来归一化每一项残差。因此公式(4-9)可以变成，cost function可以表示成残差的形式：

- $L(x) = \sum_{i=1}^m \lambda_i r_i(x)^2 = \sum_{i=1}^m \frac{1}{y_i^2}(y_i - f_i(x_i))^2$ 此时，Jacobia变成
- $H_{jk} \approx 2\sum_i J_{ij}J_{ik} \quad With \quad J_{ij} = y_i \frac{\partial r_i}{\partial x_j}$

如果我们假设数据中每一点的贡献是等权重，因此，我们有必要对每一项残差加一个权重，来归一化每一项残差。因此公式(4-9)可以变成，cost function可以表示成残差的形式：

- $L(x)=\sum_{i=1}^{m}\lambda_i r_i(x)^2=\sum_{i=1}^{m}\frac{1}{y_i^2}(y_i-f_i(x_i))^2$ 此时，Jacobia变成
- $H_{jk}\approx 2\sum_i J_{ij}J_{ik}\quad With\quad J_{ij}=y_i\frac{\partial r_i}{\partial x_j}$
- $J_S=\Sigma J,\Sigma_{ij}=\frac{1}{y_i}\delta_{ij}$

L2正则化与Levenberg-Marquardt算法

我们在cost fucntion加上L2正则项

- $L(\beta)=\sum_{i=1}^{m}[y_i-f(x_i,\beta)]^2+\lambda||\beta||^2$

可以表示成：

- $L(\beta)=L(\beta_0)+(\beta-\beta_0)^T\nabla_\beta L(\beta_0)+\frac{1}{2}(\beta-\beta_0)^T H(\beta-\beta_0)+\lambda||\beta||^2+O(\beta^3)$

求一阶导数：

- $L(\beta)=L(\beta_0)+(\beta-\beta_0)^T g+\frac{1}{2}(\beta-\beta_0)^T H(\beta-\beta_0)+\frac{1}{2}\lambda||\beta||^2+O(\beta^3)$

令其为0：

- $\frac{\partial L(\beta)}{\partial\beta}=g+H(\beta-\beta_0)+\lambda\beta=0$

就得到：

- $\beta=(H+\lambda I_n)^{-1}H\beta_0-(H+\lambda I_n)^{-1}g,$

第八章 机器学习

机器学习

1. 过拟合与欠拟合，交差验证的目的，超参数搜索方法，EarlyStopping
2. L1正则和L2正则的做法，正则化背后的思想，BatchNorm，Covariance Shift, L1正则产生稀疏解的原理
3. 逻辑回归为何是线性模型，LR如何解决低维不可分，从图模型角度看LR，
4. 和朴素贝叶斯和无监督
5. 几种参数估计方法MLE，MAP，贝叶斯的联系与区别
6. 简单说下SVM的支持向量，KKT，何为对偶，核的通俗理解
7. GBDT,随机森林能否并行，问问bagging, boosting
8. 生成模型，判别模型举个例子
9. 聚类方法的掌握，问问kmeans的EM推导思路，谱聚类和Grapg-cut的理解
10. 梯度下降类方法和牛顿类方法的区别，随便问问Adam，L-BFGS的思路
11. 半监督的思想，问问一些特定半监督算法是如何利用无标签数据的，从MAP角度看半监督
12. 常见的分类模型的评价指标(顺便问问交叉熵，ROC如何绘制，AUC的物理含义，类别不均匀样本)

神经网络

1. CNN中卷积操作和卷积核作用，maxpooling作用
2. 卷积层与全连接层的联系
3. 梯度爆炸与消失的概念(顺便问问神经网络权重初始化的方法，为何能减缓梯度爆炸与消失，CNN中有哪些解决方法，LSTM如何解决的，如何梯度裁剪，dropout如何用在RNN系列网络中，dropout如何防止过拟合)
4. 为何卷积可以用在图像，语音，语句上，顺便问问channel在不同类型数据源中的含义

自然语言处理，推荐系统

1. CRF跟逻辑回归，最大熵模型的关系
2. CRF的优化方法，CRF和MRF的联系，HMM与CRF的关系(顺便问问朴素贝叶斯和HMM的联系，LSTM+CRF用于序列标注的原理，CRF的点函数和边函数，CRF的经验分布)
3. wordEmbedding的几种常用方法和原理(language model,perplexity评价指标，word2vec跟Glove的异同)
4. Topic model说一说，为何CNN能用在文本分类，syntactic 和semantic问题举例，
5. 常见的sentence embedding方法，注意力机制(注意力机制的几种不同情形，为何引入，seq2seq原理)
6. 序列标注的评价指标，语义消歧的做法，常见的跟word有关的特征
7. factorizatuon machine,常见矩阵分解模型
8. 如何把分类模型用于商品推荐(包括数据集划分，模型验证等)
9. 序列学习，wide & deep model(顺便问问为何wide和deep)

第一步(一个月)，公式推导必须会，核心思想会，并且要融会贯通。

1. SVM
2. LR, LTR
3. 决策树，RF，GBDT，xgboost
4. 贝叶斯
5. 降维：PCA, SVD
6. BP的公式推导，以及写一个简单的神经网络，并跑个小的数据。用python,numpy。
7. LeetCode上面刷Easy与Medium的题
8. 看面经，找面试常遇到的问题

第二步是最优化总结(半个月)，总结常用的优化算法，比如梯度下降，牛顿，拟牛顿，trust region，二次规划。

机器学习score函数设置：

1. 回归，分类，决策树，深度神经网络，图模型，概率统计，最优化方法
2. 分类，聚类，特征选择，降维等数据挖掘技术
3. 机器学习，概率统计，最优化
4. GBDT, LR, LTR, 特征提取
5. 推荐系统基本算法: LR, GBDT, SVD/SVD++, FM/FFM具体实用场景与变形
6. 对分类，回归，聚类，标注等统计机器学习问题有深入研究，熟悉常用模型：LR，KNN,Naive bayer, rf, GBDT, SVM, PCA, SVD, kmeans, kmodes等
7. 精通主流机器学习算法，对贝叶斯，随机森林，SVM，神经网络，聚类，PCA等有深入研究
8. 熟悉数据分析思路，熟悉经典的数据挖掘，机器学习算法，如：LR，决策树，BP神经网络，SVM等浅层学习，熟悉集成算法，诸如bagging, boosting,了解深度学习CNN，RNN, 熟悉使用python,了解pandas, sklearn, xgboost
9. 熟悉常用的最优化算法设计与实现，对于非凸优化问题有深入的理解（并行模拟退火，蒙特卡洛等优化方法）
10. 机器学习算法：贝叶斯，聚类，逻辑回归，SVM，GBDT，RF

总结：

1. 分类：
    i. SVM,
    ii. LR
2. 回归：
3. 集成学习：
    i. bagging
    ii. Boosting
4. 工具
    i. sklearn, xgboost, tensorflow

分类总结：

1. Regression:
2.
    i. Ordinary Least Squares Regression(OLSR)
    ii. Linear Regression
    iii. Logistic Regression
    iv. Stepwise Regression
    v. Multivariate Adaptive Regression Spilines
    vi. Locally Estimated Scatterplot Smoothing
3. Regularization Algorithms
4.
    i. Ridge Regression
    ii. Least Absolute Shrinkage and Selection Operator(LASSO)
    iii. Least-Angle Regression(LARS)
5. Ensemble Algorithms
6.
    i. Boosting
    ii. Booststrapped Aggregation(Bagging)
    iii. Adaboost
    iv. Stacked Generalization(Blending)
    v. Gradient Boosting Machines(GBM)
    vi. Gradient Boosted Regression Trees(GBRT)
    vii. Random Forest
7. Decision Tree Algorithms
8.
    i. Classification and Regression Tree(CART)
    ii. Iterative Dichotomiser3(ID3)
    iii. C4.5 and C5.0
    iv. Chi-Squared Automatic Iteraction Detection(CHAID)
    v. Decision Stump
    vi. M5
    vii. Conditional Decision Trees
9. Dimensionality Reduction Algorithms
10.
    i. Principle Component Analysis(PCA)
    ii. Principle Component Regression(PCR)
    iii. Sammon Mapping
    iv. Multidimensional Scaling(MDS)
    v. Projection Pursuit
    vi. Linear Discriminant Analysis(LDA)
    vii. Mixture Discriminant Analysis(MDA)

viii. Quadratic Discriminant Analysis(QDA)
ix. Flexible Discriminant Analysis(FDA)

11. Bayesian Algorithms
12.
    i. Naive Bayes
    ii. Gaussian Naive Bayes
    iii. Multinomial Naive Bayes
    iv. Averaged One-Dependence Estimators(AODE)
    v. Bayesian Belief Network(BBN)
    vi. Bayesian Network(BN)
13. Clustering Algorithms
14.
    i. K-Means
    ii. K-Medians
    iii. Expectation Maximisation(EM)
    iv. Hierarchical
15. Instance-Based Algorithms
16.
    i. K-Nearest Neighbor(KNN)
    ii. Learning Vector Quantization(LVQ)
    iii. Self-Organizing Map(SOM)
    iv. Locally Weighted Learning(LWL)
17. Graphical Models
18.
    i. Bayesian Network
    ii. Markov random field
    iii. Chain Graphs
    iv. Ancestral Graph
19. Association Rule Learning Algorithms
20.
    i. Apriori Algorithm
    ii. Eclat Algorithm
    iii. FP-growth

1. Deep Learning
2.
    i. Deep Boltzmann Machine(DBM)
    ii. Deep Belief Networks(DBN)
    iii. Convolutional Neural Network(CNN)
    iv. Stacked Auto-Encoders
3. Artificial Neural Network

4.

i. Perception
ii. Back-Propagation
iii. Radial Basis Function Network(RBFN)

SVM，KKT条件与核函数方法

SVM

KKT条件

核函数方法

决策树与集成学习

决策树与集成学习

决策树是一种基本的回归与分类的方法。决策树由节点与边构成，节点分类内部节点与叶子节点；内部节点表示属性或者特征，叶子节点表示一个类。

决策树学习

训练集$D=((x_1,y_1),(x_2,y_2)...(x_N,y_N))$其中$x_i=(x_i^{(1)},x_i^{(2)}...x_i^{(n)})$是n维变量，n表示特征的数目，$y_i\in(1,2...K)$类标签。训练的本质是基于一定标准得到一组分类规则。我们需要得到一组与训练数据矛盾较小并且泛化能力也好的分类规则，其中泛化能力说的是在测试集上其错误率也小，错误率有不同的定义方式，可以是均方误差，也可以是似然函数，一般回归问题选择均方误差，分类问题选择似然函数，想想这是为什么？。
分类规则就是特征选择的过程，特征选择是基于一些可以量化的函数，一般基于信息熵增益最大或者信息熵增益率最大或者Gini系数下降最大的规则。因此，接下来我们要定义如下概念。

1. 信息熵
2. 条件熵
3. 信息增益
4. 信息增益率
5. Gini系数

1. 信息熵

熵的概念来自于统计物理，描述微观系统的混乱程度，由Rudolf Clausius提出，

$$S=k_B\ln\Omega=-k_B\sum_i p_i\ln p_i$$

其中k_B是Boltzmann constant,Ω表示系统的可能状态数。

它表示的就是$ln(p_i)$的期望值。香农把这个概念引入信息学，定义了信息熵。
考虑一个只能取有限n个值的系统，值对应的就是状态，比如投硬币，只能取两个值，就是说只能有2个状态，因此n=2，在统计物理中，状态对应的是系统的能级，也就是一个能级对应一个状态，系统处于不同能级的概率就是p_i

$$P(X=x_i)=p_i,i=1,2..n$$

熵的定义如下：$H(x)=-\sum_i^{i=n}p_i\ln p_i$

若系统是确定的，则有一个$p_i=1$，其他的$p_i=0$等于0，因此代入熵的公式可知，熵为0.可以证明，对于n个状态的系统，系统的熵满足如下不等式：

熵的概念来自于统计物理，描述微观系统的混乱程度，由Rudolf Clausius提出，

$$S = k_B \ln \Omega = -k_B \sum_i p_i \ln p_i$$

其中k_B是Boltzmann constant,Ω表示系统的可能状态数。

它表示的就是$ln(p_i)$的期望值。香农把这个概念引入信息学，定义了信息熵。
考虑一个只能取有限n个值的系统，值对应的就是状态，比如投硬币，只能取两个值，就是说只能有2个状态，因此n=2，在统计物理中，状态对应的是系统的能级，也就是一个能级对应一个状态，系统处于不同能级的概率就是p_i

$$P(X = x_i) = p_i, i = 1, 2..n$$

熵的定义如下：$H(x) = -\sum_i^{i=n} p_i \ln p_i$

若系统是确定的，则有一个$p_i = 1$，其他的$p_i = 0$等于0，因此代入熵的公式可知，熵为0.可以证明，对于n个状态的系统，系统的熵满足如下不等式：

$$0 \leq H(x) = -\sum_i^{i=n} p_i ln p_i \leq \ln n.$$

2. 条件熵

联合概率说的是两个及两个系统随机变量共同发生的几率问题，条件熵$H(Y|X)$说的是随机变量X给定的情况下，随机变量Y的条件熵。
计算如下：

$$H(Y|X) = \sum_i^{i=n} p_i H(Y|X = x_i)$$

这里，$p_i = P(X = x_i), i = 1, 2..n$。

$H(Y|X = x_i)$计算与上面的信息熵一样，只是对于$H(Y|X = x_i)$，我们只计算$X = x_i$的那些样本的熵，因为我们会对所有X取不同值得样本进行求和，因此会遍历整个样本。

3. 信息增益

信息增益是基于以上两个概念的，是针对于特征而言的，特征A对训练数据集D的信息增益g(D,A)定义成经验熵H(D)与特征A给定的条件下，D的经验条件熵H(D|A)之差。即：

$$g(D, A) = H(D) - H(D|A)。$$

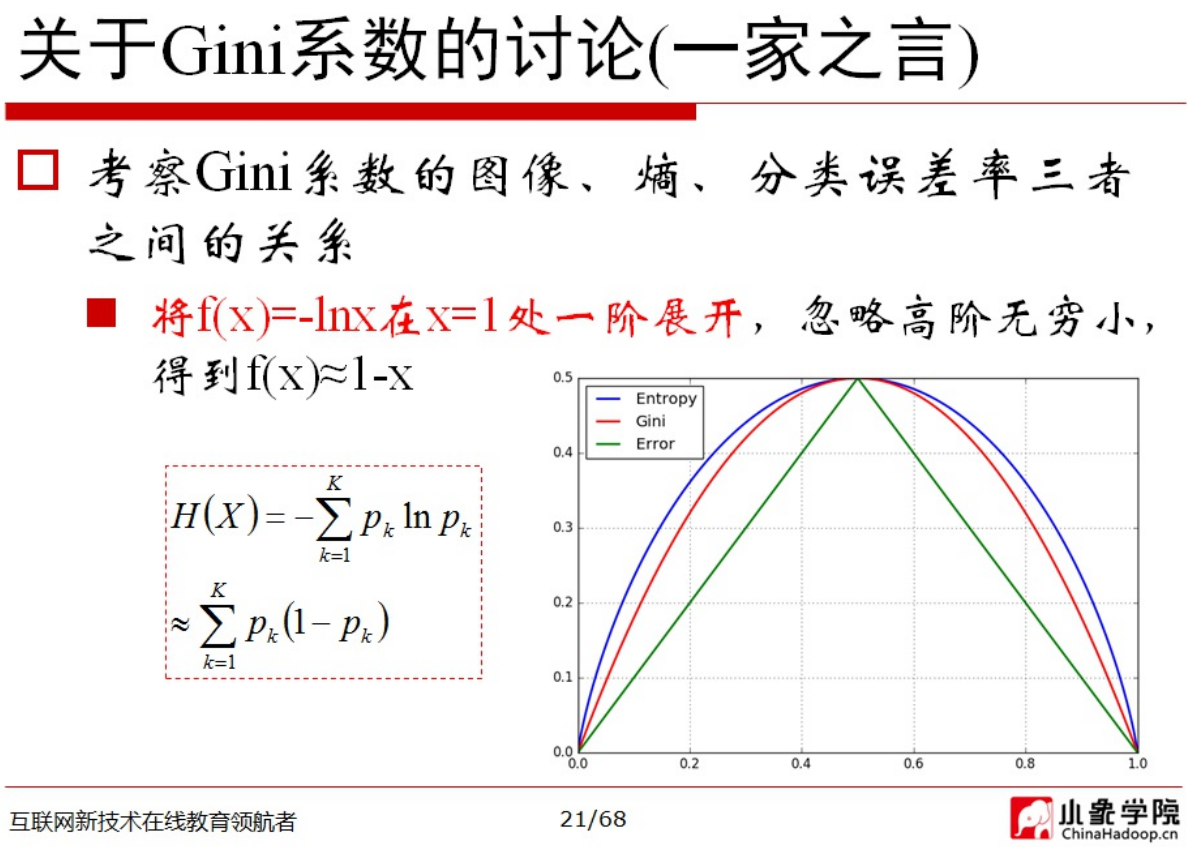
如果我们知道数据集D的信息熵计算，以及条件熵的计算，则信息增益的计算不难。

4. 信息增益率

假设样本D中有K个类，C_k是第k类的集合，$|C_k|$是集合C_k的大小，因此样本集合的基尼系数定义成:

$$Gini(p) = 1 - \sum_{i=1}^{i=K} (\frac{|C_k|}{|D|})^2$$

基于不同的指标，比如信息增益，信息增益率，基尼系数，我们会得到不同的决策树生成算法，依次是ID3，C4.5, CART。



信息熵与基尼系数的函数大致一致，实际上在iris数据集中，生成决策树时，两者没有差别。

ID3算法

ID3算法基于信息增益最大，代码实现如下，它的缺点在于，它倾向于选择取值很多的特征，因为当特征能取很多值得时候，此时系统的不确定度降低，极端情况是，特征能取N个不同的值，N个训练集的数量，此时特征A条件熵为0. 为了避免这种情况，而引入了C4.5算法。

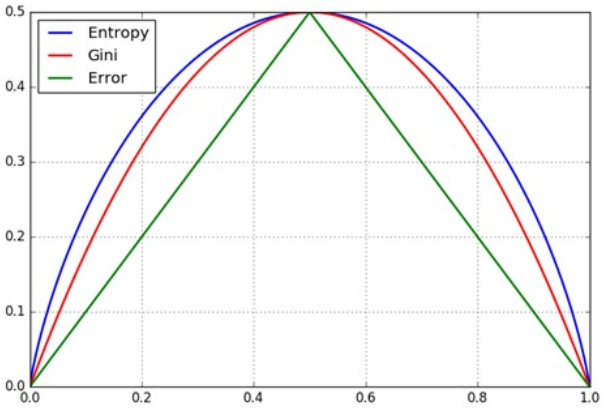
假设特征A是信息增益最大的特征，当特征A取离散值时，假设特征A可以取K个不同值，我们选择A作为特征之后会生出K个节点，对于每个节点，我们要重复上面的步骤，继续寻找信息增益最大的特征，只是，这时候特征数目减小了1，减小的这个特征就是A特征本身。当一个节点，它的熵小于一定阈值的时候，我们就不再分割这个节点而是把他当成一个叶子节点。我们可以使用递归来

关于Gini系数的讨论(一家之言)

- 考察Gini系数的图像、熵、分类误差率三者之间的关系
  - 将f(x)=-lnx在x=1处一阶展开，忽略高阶无穷小，得到f(x)≈1-x

$$H(X) = -\sum_{k=1}^{K} p_k \ln p_k$$

$$\approx \sum_{k=1}^{K} p_k (1 - p_k)$$



信息熵与基尼系数的函数大致一致，实际上在iris数据集中，生成决策树时，两者没有差别。

ID3算法

ID3算法基于信息增益最大，代码实现如下，它的缺点在于，它倾向于选择取值很多的特征，因为当特征能取很多值得时候，此时系统的不确定度降低，极端情况是，特征能取N个不同的值，N个训练集的数量，此时特征A条件熵为0. 为了避免这种情况，而引入了C4.5算法。

假设特征A是信息增益最大的特征，当特征A取离散值时，假设特征A可以取K个不同值，我们选择A作为特征之后会生出K个节点，对于每个节点，我们要重复上面的步骤，继续寻找信息增益最大的特征，只是，这时候特征数目减小了1，减小的这个特征就是A特征本身。当一个节点，它的熵小于一定阈值的时候，我们就不再分割这个节点而是把他当成一个叶子节点。我们可以使用递归来实现ID3算法。

对于特征值取连续的情况，我们通过计算特征值大于阈值与小于阈值的两部分熵之和来计算熵。算法实现的时候要注意的一点就是，计算熵时，不能让概率等于0，否则会出错。

```
import scipy
import numpy as np
from sklearn import tree
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

class DecisionTree(object):
```

```
            for n in range(1, sorted_data.shape[0]):
                threshold = (sorted_data[n-1, feature_index] + sorted_data[n, feature_index])/
2
                featue_result_data = np.zeros((sorted_data.shape[0], 2))
                featue_result_data[:, 0] = sorted_data[:, feature_index]
                featue_result_data[:, -1] = sorted_data[:, -1]
                condition_entropy = self.distribution(featue_result_data, n)
                if condition_entropy < best_entropy:
                    best_threshold = threshold
                    best_entropy = condition_entropy
                    best_feature = feature_index
            print best_entropy, best_feature, best_threshold
        return best_entropy, best_feature, best_threshold

if __name__ == '__main__':
    DT = DecisionTree()
    iris = load_iris()
    clf = tree.DecisionTreeClassifier()
    clf = clf.fit(iris.data, iris.target)
    data = iris.data
    target = iris.target
    iris_data = np.zeros((data.shape[0], data.shape[1] + 1))
    iris_data[:, 0:data.shape[1]] = data
    iris_data[:, -1] = target
    best_entropy, best_feature, best_threshold = DT.best_feature_threshold(iris_data)
```

C4.5

C4.5算法弥补了ID3算法的不足，通过使用信息增益率来作为特征的选择标准。

- $g_R(D, A) = \frac{g(D,A)}{H(A}$。

其中$H(A) = \sum_{i=1}^{i=K} p_i \ln p_i$,K是特征A能取不同值得数目，$p_i$是取不同值得概率。

模型的剪枝

为什么需要剪枝：提高泛化能力，
我们可以一步步的细分节点而使得系统的分类误差很小，但是这只是训练数据集上的结果，当我们把模型运用到测试集时，误差率会很高，这是因为模型过拟合你，我们可以通过对决策树进行剪枝来提高模型的泛化能力。
为了进行剪枝，我们得定义剪枝的标准，也就是定义损失函数，损失函数至少要包括两项，一个是模型在训练集上的误差，一个是模型的复杂程度。模型的复杂程度可以定义成与叶子节点正相关。模型在训练集上的误差率可以如下定义。

- $C(T) = \sum_{t=1}^{t=|T|} N_t H_t$

```
        return best_entropy, best_feature, best_threshold

if __name__ == '__main__':
    DT = DecisionTree()
    iris = load_iris()
    clf = tree.DecisionTreeClassifier()
    clf = clf.fit(iris.data, iris.target)
    data = iris.data
    target = iris.target
    iris_data = np.zeros((data.shape[0], data.shape[1] + 1))
    iris_data[:, 0:data.shape[1]] = data
    iris_data[:, -1] = target
    best_entropy, best_feature, best_threshold = DT.best_feature_threshold(iris_data)
```

C4.5

C4.5算法弥补了ID3算法的不足，通过使用信息增益率来作为特征的选择标准。

$$g_R(D,A)=\frac{g(D,A)}{H(A}。$$

其中$H(A)=\sum_{i=1}^{i=K}p_i\ln p_i$,K是特征A能取不同值得数目，$p_i$是取不同值得概率。

模型的剪枝

为什么需要剪枝：提高泛化能力，
我们可以一步步的细分节点而使得系统的分类误差很小，但是这只是训练数据集上的结果，当我们把模型运用到测试集时，误差率会很高，这是因为模型过拟合你，我们可以通过对决策树进行剪枝来提高模型的泛化能力。
为了进行剪枝，我们得定义剪枝的标准，也就是定义损失函数，损失函数至少要包括两项，一个是模型在训练集上的误差，一个是模型的复杂程度。模型的复杂程度可以定义成与叶子节点正相关。模型在训练集上的误差率可以如下定义。

$$C(T)=\sum_{t=1}^{t=|T|}N_tH_t$$

其中N_t是叶子节点t上的样本点数目，H_t是叶子节点t的熵。
叶子节点的熵定义为：

$$H_t=-\sum_{k=1}^{k=K}\frac{N_{tk}}{N_t}\ln\frac{N_{tk}}{N_t},$$

其中$k\in(1,2...K)$是样本结果可以取不同值的数目，也就是标签有K类N_t是叶子节点t的样本数目，N_{tk}是叶子节点t中标签属于k类的数目。总体而言，C(T)刻画的是模型在训练集上的误差。结合这两部分，我们可以定义剪枝的损失函数：

$$C_\alpha(T)=C(T)+\alpha|T|$$

假设X和Y分别为输入与输出变量，并且Y是连续变量，给定训练数据集

$$D = ((x_1, y_1), (x_2, y_2)...(x_N, y_N)).$$

假设已将输入空间划分为M个单元$R_1, R_2, ..., R_M$,并且每个单元R_m上有一个固定的输出值c_m,于是回归树模型可以表示为:

$$f(x) = \sum_{m=1}^{m=M} c_m I(x \in R_m)$$

当输入空间的划分确定时，可以用平方误差

$$\sum_{x_i \in R_m} (y_i - f(x_i))^2$$

来表示回归书对于训练数据的预测误差，用平方误差最小的准则来求解每个单元上的最优输出值。

可以求得

$$\hat{c}_m = ave(y_i | x_i \in R_m)。$$

问题是，怎么对输入的空间进行划分？

回归树可以处理离散特征与连续特征，对于连续特征，若这里按第j个特征的取值s进行划分，切分后的两个区域分别为:

$$R_1(j, s) = (x_i | x_i^j \leq s)$$

$$R_2(j, s) = (x_i | x_i^j > s)$$

若为离散特征，则找到第j个特征下的取值s:

$$R_1(j, s) = (x_i | x_i^j = s)$$

$$R_2(j, s) = (x_i | x_i^j \neq s)$$

分别计算R_1, R_2的类别估计c_1, c_2,然后计算按照(j,s)切分后的损失:

$$min_{j,s}[\sum_{x_i \in R_1} (y_i - c_1)^2 + \sum_{x_i \in R_2} (y_i - c_2)^2]$$,找到是的损失最小的(j,s)对

即可，也就是说找到最优特征j^*，并找到最优特征的分割值s^*s。递归执行(j,s)的选择过程，知道满足停止条件为止。递归的意思是，对分割出的两个区域R_1, R_2分别进行如上的步骤，再分割下去。

总结:

回归树算法:

输入:训练集 $D = ((x_1, y_1), (x_2, y_2)...(x_N, y_N)).$

输出:回归树T

1.求解选择的切分特征j与切分特征取值s, j将训练集D划分成两部分， R_1, R_2,

$$R_1(j, s) = (x_i | x_i^j \leq s)$$

$$R_2(j, s) = (x_i | x_i^j > s)$$

其中:

- $\hat{c}_m = ave(y_i|x_i \in R_m)$。

问题是，怎么对输入的空间进行划分？

回归树可以处理离散特征与连续特征，对于连续特征，若这里按第j个特征的取值s进行划分，切分后的两个区域分别为：

- $R_1(j,s) = (x_i|x_i^j \leq s)$
- $R_2(j,s) = (x_i|x_i^j > s)$

若为离散特征，则找到第j个特征下的取值s:

- $R_1(j,s) = (x_i|x_i^j = s)$
- $R_2(j,s) = (x_i|x_i^j \neq s)$

分别计算R_1, R_2的类别估计c_1, c_2,然后计算按照(j,s)切分后的损失:

- $min_{j,s}[\sum_{x_i \in R_1}(y_i - c_1)^2 + \sum_{x_i \in R_2}(y_i - c_2)^2]$,找到是的损失最小的(j,s)对

即可，也就是说找到最优特征j^*，并找到最优特征的分割值s^*s。递归执行(j,s)的选择过程，知道满足停止条件为止。递归的意思是，对分割出的两个区域R_1, R_2分别进行如上的步骤，再分割下去。

总结：

回归树算法：

输入：训练集 $D = ((x_1, y_1), (x_2, y_2)...(x_N, y_N))$.

输出：回归树T

1.求解选择的切分特征j与切分特征取值s, j将训练集D划分成两部分， R_1, R_2,

- $R_1(j,s) = (x_i|x_i^j \leq s)$
- $R_2(j,s) = (x_i|x_i^j > s)$

其中：

- $c_1 = \frac{1}{N_1}\sum_{x_i \in R_1} y_i$
- $c_2 = \frac{1}{N_2}\sum_{x_i \in R_2} y_i$

2.遍历所有的可能解(j,s),找到最优的(j^*, s^*),最优解使得如下损失函数最小，

- $min_{j,s}[\sum_{x_i \in R_1}(y_i - c_1)^2 + \sum_{x_i \in R_2}(y_i - c_2)^2]$

按照最优特征(j^*, s^*)来切分即可

3.递归调用1., 2.步骤，直到满足停止条件

4.将输入空间划分为M个区域$R_1, R_2, ..., R_M$,返回决策树T

- $f(x) = \sum_{m=1}^{m=M} \hat{c}_m I(x \in R_m)$

对于固定的α，存在唯一的最有子树$C_\alpha(T)$.

Breiman等人证明，可以用递归的方法对树进行剪枝，将α从0开始增大，$0 \leq \alpha_1 \leq \alpha_2 \leq ... \leq \alpha_n$，

得到一系列最优子树系列 (T_0, T_1, T_n),序列中的子树是嵌套的（？）。

具体的，从整棵树T_0开始剪枝，对T_0的任意内部节点t,以t为单节点书的损失函数是(也就是把t下面的所有节点减掉之后的损失函数)：

$$C_\alpha(t) = C(t) + \alpha$$

以t为根结点的字数T_t的损失函数是(不进行剪枝时的损失函数)：

$$C_\alpha(T_t) = C(T_t) + \alpha|T_t|$$

当\alpha 小于某个值时，不剪枝会使得整体的损失函数较小，也就是：

$$C_\alpha(T_t) < C_\alpha(t)$$

当α增大时，在某一α有：

$$C_\alpha(T_t) = C_\alpha(t)$$

当α再增大时，不等式反向，只要$\alpha = \frac{C(t)-C(T_t)}{|T_t|-1}$, T_t与t有相同的损失函数。

为此，对T_0中每一个内部节点t,计算：

$$g(t) = \frac{C(t)-C(T_t)}{|T_t|-1}$$

它表示剪枝之后整体损失函数减小的程度，在T_0中减去$g(t)$最小的T_t,将得到的子树作为T_1,同时

将最小的g(t)设为α_1, T_1为区间$[\alpha_1, \alpha_2)$的最优子树。
如此剪枝下去，在这一过程中，不断的增加α的取值，产生新的区间，最终会的到一组决策树

(T_0, T_1, T_n)，对应于椅子确定的权衡参数(a_0, a_1, a_n)，通过验证集合中每颗树的总体误差，也就得

到了最终的最优决策树T^*.

T是整颗决策树吗？

输入：CART 生成树 T_0

输出：剪枝后的最优树 T^*

1）设 k=0，$T = T_0$，$a = +\infty$

3) 自下而上的对内部节点 t 计算：

$C(t)-C(T)$

当\alpha 小于某个值时，不剪枝会使得整体的损失函数较小，也就是：

$$C_\alpha(T_t) < C_\alpha(t)$$

当α增大时，在某一α有：

$$C_\alpha(T_t) = C_\alpha(t)$$

当α再增大时，不等式反向，只要$\alpha = \frac{C(t)-C(T_t)}{|T_t|-1}$，$T_t$与t有相同的损失函数。

为此，对T_0中每一个内部节点t,计算：

$$g(t) = \frac{C(t)-C(T_t)}{|T_t|-1}$$

它表示剪枝之后整体损失函数减小的程度，在T_0中减去$g(t)$最小的T_t,将得到的子树作为T_1,同时将最小的g(t)设为α_1，T_1为区间$[\alpha_1, \alpha_2)$的最优子树。

如此剪枝下去，在这一过程中，不断的增加α的取值，产生新的区间，最终会的到一组决策树(T_0, T_1, T_n)，对应于椅子确定的权衡参数(a_0, a_1, a_n)，通过验证集合中每颗树的总体误差，也就得到了最终的最优决策树T^*.

T是整颗决策树吗？

输入：CART 生成树 T_0

输出：剪枝后的最优树 T^*

1）设 k=0，$T = T_0$，$a = +\infty$

3) 自下而上的对内部节点 t 计算：

$$g(t) = \frac{C(t)-C(T_t)}{|T_t|-1}$$

a=min(a,g(t))

4) 自上而下的访问内部节点 t，对最小的 g(t)=a 进行剪枝，并对叶节点 tt 以多数表决形式决定其类别，得到树 TT

5) k=k+1,$a_k = a$, $T_k = T$

6) 如果 T 为非单节点树，回到 4).

7) 对于产生的子树序列 (T_0, T_1, T_n) 分别计算损失，得到最优子树T^*并返回.

使得这两部分的误差和最小，这就是该特征的最佳切分点，再在该数据集中的所有特征中这个误差，找到最佳的切分特征，最后得到最佳特征以及最佳特征的切分点。基于这两个值，把数据集分成两部分，再对这两部分分别进行如上的操作（求最佳特征与该特征下的最佳切分点）

[1]. 决策树之 CART . ↩

集成学习

集成学习就是组合一系列的弱分类器生成强分类器。组合的弱分类器之间的关系有两种:

一种是彼此间相互依赖,一系列学习器之间需要串行生成，代表算法是Boosting系列算法，代表算法是AdaBoost与GBDT.
一种是学习器彼此间无关联，一些列算法可以并行的生成，代表算法是Bagging(Bootstrap Aggregating)和随机森林（Random Forest）系列。随机森林是时Bagging的进行版，随机表现在两个方面，一方面是取样本点是随机的，另一方面，选择特征也是随机的（比如总共有N个特征，随机选取小于N的K个特征）
还有一种是两者的结合物：stacking

PAC，弱可学习，强可学习

PAC(Probably approximately correct)
强可学习：如果存在一个多项式的学习算法学习呀，学习的正确率很高
弱可学习：如果存在一个多项式的学习算法学习呀，学习的正确率仅比随机猜测略好
在PAC学习框架下，一个概念是强可学习的充分必要条件是这个概念是弱可学习的。

集成学习的理论基础

我们考虑多个不相干分类器叠加处理二分类的问题，$y \in (-1,+1)$和真实的函数f,假设基分类器的错误率都是ϵ, 即对每个分类器都有

$$P(h_i(x) \neq f(x)) = \epsilon$$

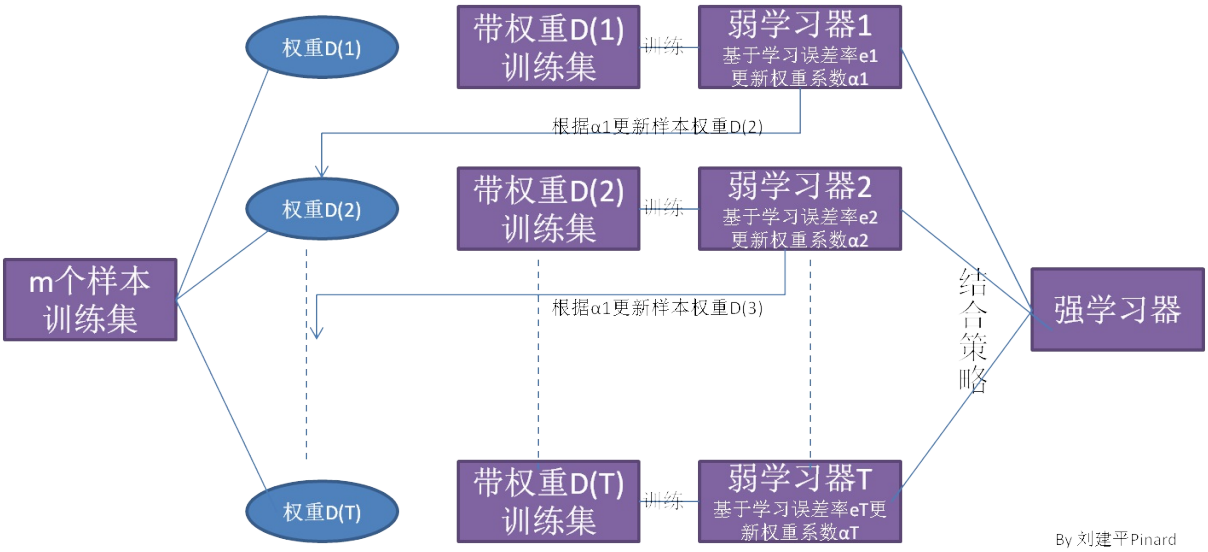
假设集成通过简单的投票发结合T个基分类器，若半数的基分类器正确，则集成分类正确。
{% math %}\kern{10 em}H(x) = sign(\sum _{i=1}^{T}h_i(x)){% endmath %}
价格基分类器之间的错误率是相互独立的，则由Hoeffding不等式可知，集成的错误率为：

$$P(H(x) \neq f(x)) = \sum_{i=0}^{[T/2]} \binom{T}{k} (1-\epsilon)^k \epsilon^{T-k} \leq exp(-\frac{1}{2}T(1-2\epsilon)^2)$$

上式表明，随着集成中个体分类器数目T的增大，集成的错误率将指数下降，最终趋向于0.

Boosting:

Boosting系列方法的原理图如下：



AdaBoost

Boosting系列算法的代表就是AdaBoost。

算法如下：

输入：训练数据集 $T=((x_1,y_1),(x_2,y_2)...(x_N,y_N))$,其中$x_i \in X \subseteq R^n, y \in (-1,+1)$

输出：最终的分类器G(x)

(1)初始化训练数据集的权重分布

$$D_1=(w_{11},...,w_{1i},...,w_{1N}),$$

$$w_{1i}=\frac{1}{N},i=1,2,...,N$$

(2)对m=1,2,...,M

使用具有权值分布D_m的训练数据集进行训练，得到基分类器

$$G_m(x):\chi \in (-1,+1)$$

(b)计算$G_m(x)G_m(x)$在训练集上的分类误差率：

$$e_m=P(G_m(x_i)\neq y_i)=\sum_{i=1}^{N}w_{mi}I(G_m(x_i)\neq y_i)$$

(c)计算$G_m(x)$的系数

$$\alpha_m=\frac{1}{2}\log\frac{1-e_m}{e_m}$$

这里用的是自然对数。

(d)更新训练数据集的权值分布：

$$D_1=(w_{m+1,1},...,w_{m+1,i},...,w_{m+1,N}),$$

$$w_{m+1,i}=\frac{w_{mi}}{Z_m}exp(-\alpha_m y_i G_m(x_i)),i=1,2,...,N$$

这里，Z_m是归一化因子：

$$Z_m = \sum_{i=1}^{N} w_{mi} exp(-\alpha_m y_i G_m(x_i))$$

它使D_{m+1}成为一个概率分布。

(3)构建基本分类器的线性组合:

$$f(x) = \sum_{i=1}^{N} \alpha_m G_m(x)$$

得到最终分类器

$$G(x) = sign(f(x)) = sign(\sum_{i=1}^{N} \alpha_m G_m(x))$$

由α_m的定义可以知道，第m个分类器的误差率越小，则α_m的值越大，因此第m个分类器在总的分类器中占的比重越大。因此，AdaBoost会加大那些准确率很高的分类器的权重。

同时，在第m+1个分类器求解时，对于上一轮被分错的样本的权值会变大。

不改变所给的训练数据，而不断的改变训练数据权值的分布，使得训练数据在基本分类器的学习中起不同的作用，这是AdaBoost的一个特点。

可以参考《统计机器学习》8.1节的例子来加深理解。

AdaBoost算法的训练误差分析

定理:**(AdaBoost的训练误差界)**

AdaBoost 算法最终分类器的训练误差界为:

$$\frac{1}{N} \sum_{i=1}^{N} I(G_m(x_i) \neq y_i) \leq \frac{1}{N} \sum_{i=1}^{N} exp(-y_i f(x_i)) = \prod_{m} Z_m$$

定理**(二分类问题AdaBoost的训练误差界)**

$$\prod_{m} Z\_m = \prod_{m=1}^{M} [2\sqrt{e_m(1-e_m)}] = \prod_{m=1}^{M} \sqrt{1-4\gamma_m^2} \leq exp(-2\sum_{m=1}^{M} \gamma\_m^2)$$

其中: $\gamma_m = \frac{1}{2} - e_m$

梯度提升

提升树是以分类树或者回归树为基本分类器的提升方法，提升树被认为是统计学习中性能最好的方法之一。

提升树实际采用加法模型(即基函数的线性组合)与前向分步算法。以决策树为基函数的提升方法称为提升树。对分类问题决策树是二叉分类树，对回归问题决策树是二叉回归树。

决策树桩(decsion stump):可以看成是一个根节点直接连接两个叶节点的简单决策树。提升树模型可以表示为决策树的加法模型。

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m)$$

其中:$T(x,\Theta_m)$表示决策树;Θ_m为决策树的参数;M为树的个数。

梯度提升算法:

提升树算法采用前向分步算法。首先缺点初始提升树$f_0(x)=0$,第m步的模型是:

- $f_m(x)=f_{m-1}(x)+T(x;\Theta_m)$

其中，$f_{m-1}(x)$为当前模型，通过经验风险极小化来确定下一刻决策树的参数Θ_m

- $\hat{\Theta}_m=argmin_{\Theta_m}\sum_{i=1}^{N}L(y_i,f_{m-1}(x_i)+T(x_i;\Theta_m))$.

对于一个训练数据集 $T=((x_1,y_1),(x_2,y_2),...,(x_N,y_N)),x_i\in\chi\in R^N$, χ为输入控件， $y_i\in R$,为输出空间。如果将输入控件χ划分为J个互不相交的区域$R_1,R_2,...,R_J$，并且在每个区域上确定输出的常量c_j，那么树可以表示为:

- $T(x;\theta)=\sum_{j=1}^{J}c_jI(x\in R_j)$.

其中，参数$\Theta=((R_1,c_1),(R_2,c_2),...,(R_J,c_J))$表示树的区域划分和各区域上的常数。J是回归树的发杂都，即叶节点个数。

回归为题提升树使用以下前向分步算法:

- $f_0(x)=0$
- $f_m(x)=f_{m-1}(x)+T(x;\Theta_m),m=1,2,...,M$
- $f_m(x)=\sum_{m=1}^{M}T(x;\Theta_m)$

在前向分步算法的第m步，给定当前模型$f_{m-1}(x)$，需求解:

- $\hat{\Theta}_m=argmin_{\Theta_m}\sum_{i=1}^{N}L(y_i,f_{m-1}(x_i)+T(x_i;\Theta_m))$.

得到$\hat{\Theta}_m$，即第m颗树的参数。

采用平方误差损失函数时:

- $L(y,f(x))=(y-f(x))^2$

其损失变成:

- $L(y,f_{m-1}(x)+T(x;\Theta_m))=(y-f_{m-1}(x)-T(x;\Theta_m))^2=[r-T(x;\Theta_m)]^2$

这里，

- $r=y-f_{m-1}(x)$

是当前模型拟合数据的残差。所以，对回归问题的提升树算法来说，只需要简单地你和当前模型的残差。

GBDT

当损失函数是平方损失和指数损失函数时，每一步优化是很简单的，但是对于一般损失函数而言，往往每一步优化并不容易。针对这一问题，Freidman提出了梯度梯度提升(gradient boosting)算法,这是利用最速下降法的近似方法，其关键是利用损失函数的负梯度在当前模型的值。

- $-[\frac{\partial L(y,f(x_i))}{\partial f(x_i)}]_{f(x)=f_{m-1}(x)}$

作为回归问题提升算法中的残差的近似值，拟合一个回归树。

梯度提升树算法

输入：训练数据集 $T=((x_1,y_1),(x_2,y_2),...,(x_N,y_N)),x_i\in\chi\in R^N$，$\chi$为输入，$y_i\in R$，损失函数为$L(y,f(x))$:

输出：回归树 $\hat{f}(x)$.

(1)初始化

- $f_0(x)=argmin_c\sum_{i=1}^{N}L(y_i,c).$

(2)对m=1,2,...,M

(a)对i=1,2,...,N，计算

- $r_{mi}=-[\frac{\partial L(y,f(x_i))}{\partial f(x_i)}]_{f(x)=f_{m-1}(x)}$

(b)对r_{mi}拟合一个回归树，得到第m课树叶节点区域$R_{my},j=1,2,..,J$

(c)对j=1,2,..,J，计算

- $c_{mj}=argmin_c\sum_{x_i\in R_{mj}}L(y_i,f_{m-1}(x_i)+c).$

(d)更新$f_m(x)=f_{m-1}(x)+\sum_{j=1}^{J}c_{mj}I(x\in R_{mj}),m=1,2,...,M$

(3)得到回归树

- $\hat{f}(x)=f_M(x)=\sum_{m=1}^{M}\sum_{j=1}^{J}c_{mj}I(x\in R_{mj})$

函数空间的数值优化：

XGBoost[1]

模型复杂度惩罚是XGBoost相对于MART的提升。

- $=\sum_{m=1}^{M}[\gamma T_m+\frac{1}{2}\lambda||w_m||_2^2+\alpha||w_m||_1]$

MART includes row subsampling, while XGBoost includes both row and column subsampling

Newton boosting

Input: Data set D, A loss function L, A base learner L_Φ, the number of iterations M. The learning rate η

1. Initialize $\hat{f}^{(0)}(x) = \hat{f}_{(0)}(x) = \hat{\theta}_0 = argmin_\theta \sum_{i=1}^{n} L(y_i, \theta)$;
2. for m = 1,2,...,M do
3. $\hat{g}_m(x_i) = [\frac{\partial L(y, f(x_i))}{\partial f(x_i)}]_{f(x)=f^{(m-1)}(x)}$
4. $\hat{h}_m(x_i) = [\frac{\partial^2 L(y, f(x_i))}{\partial f(x_i)^2}]_{f(x)=f^{(m-1)}(x)}$
5. $\hat{\phi}_m = argmin_{\phi \in \Phi} \sum_{i=1}^{n} \frac{1}{2} \hat{h}_m(x_i)[(-\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}) - \phi(x_i)]^2$
6. $\hat{f}_m(x) = \eta \hat{\phi}_m(x)$;
7. $\hat{f}^m(x) = \hat{f}^{(m-1)}(x) + \hat{f}_m(x)$;
8. end

Output: $\hat{f}(x) = \hat{g}^{(M)}(x) = \sum_{m=1}^{M} \hat{f}_m(x)$

对于没有惩罚项的Netwon tree boosting(NTB),每一次迭代都市最小化如下损失函数:

- $J_m(\phi_m) = \sum_{i=1}^{n} L(y_i, \hat{f}^{(m-1)(x_i)} + \phi_m(x_i))$

对于梯度提升算法，基函数是如下的树:

- $\phi_m(x) = \sum_{j=1}^{T} w_{jm} I(x \in R_{jm})$

这里的T是第m颗树叶子节点的个数，w_jm是第m颗树中第j个叶子节点的权重。

对上式进行二阶泰勒展开,并忽略掉常数项可以得到:

- $J_m(\phi_m) = \sum_{i=1}^{n} [\hat{g}_m(x_i)\phi_m(x_i) + \frac{1}{2}\hat{h}_m(x_i)\phi_m(x_i)^2]$

代入:

- $\phi_m(x) = \sum_{j=1}^{T} w_{jm} I(x \in R_{jm})$
- $J_m(\phi_m) = \sum_{i=1}^{n} [\hat{g}_m(x_i) \sum_{j=1}^{T} w_{jm} I(x \in R_{jm}) + \frac{1}{2}\hat{h}_m(x_i)(\sum_{j=1}^{T} w_{jm} I(x \in R_{jm}))^2]$

由于每个样本点只能属于一个叶子节点，因此:

- $I(x \in R_{jm}) I(x \in R_{im}) = \delta_{ij}$

因此上面可以简化成:

- $J_m(\phi_m) = \sum_{i=1}^{n} [\hat{g}_m(x_i) \sum_{j=1}^{T} w_{jm} I(x \in R_{jm}) + \frac{1}{2}\hat{h}_m(x_i) \sum_{j=1}^{T} w_{jm}^2 I(x \in R_{jm})]$
- $= \sum_{j=1}^{T} \sum_{i \in I_{jm}} [\hat{g}_m(x_i) w_{jm} + \frac{1}{2}\hat{h}_m(x_i) w_{jm}^2]$

定义:

- $G_{jm} = \sum_{i \in I_{jm}} \hat{g}_m(x_i)$

- $H_{jm} = \sum_{i \in I_{jm}} \hat{h}_m(x_i)$

因此，可以把cost function写成：

$$J_m(\phi_m) = \sum_{j=1}^{T}[G_{jm}w_{jm} + \frac{1}{2}H_{jm}w_{jm}^2]$$

- $\geq -\sum_{j=1}^{T}\frac{1}{2}\frac{G_{jm}^2}{H_{jm}}$

成立条件是：

$w_{jm} = -\frac{G_{jm}}{H_{jm}}, j = 1, 2..., T$

为了寻找最佳分裂点j，也就是最大化如下的Gain:

- $Gain = \frac{1}{2}[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G_{jm}^2}{H_{jm}}]$

总结起来：Newton tree boosting算法如下：

Input: Data set D, A loss function L, A base learner L_Φ, the number of iterations M. The learning rate η

1. Initialize $\hat{f}^{(0)}(x) = \hat{f}_{(0)}(x) = \hat{\theta}_0 = argmin_\theta \sum_{i=1}^{n} L(y_i, \theta)$;
2. for m = 1,2,...,M do
3. $\hat{g}_m(x_i) = [\frac{\partial L(y, f(x_i))}{\partial f(x_i)}]_{f(x)=f^{(m-1)}(x)}$
4. $\hat{h}_m(x_i) = [\frac{\partial^2 L(y, f(x_i))}{\partial f(x_i)^2}]_{f(x)=f^{(m-1)}(x)}$
5. Determin the structure $\hat{R}_{jm}, j = 1, ..., T$ by selecting splits which maximize

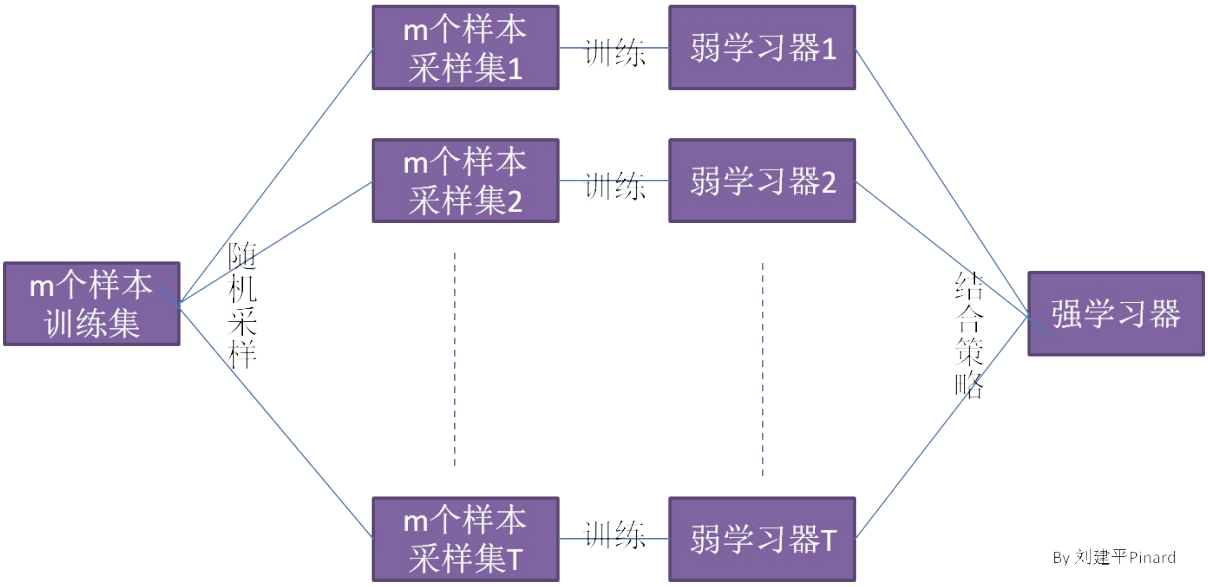
   - $Gain = \frac{1}{2}[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G_{jm}^2}{H_{jm}}]$
6. Determine the leaf weights $w_{jm}, j = 1, ..., T$ for the learnt structure by

   $\hat{w}_{jm} = -\frac{G_{jm}}{H_{jm}}, j = 1, 2..., T$
7. $\hat{f}_m(x) = \eta \sum_{j=1}^{T} \hat{w}_{jm} I(x \in \hat{R}_{jm})$;
8. $\hat{f}^m(x) = \hat{f}^{(m-1)}(x) + \hat{f}_m(x)$;
9. end

   Output: $\hat{f}(x) = \hat{g}^{(M)}(x) = \sum_{m=1}^{M} \hat{f}_m(x)$

Bagging

Bagging系列方法的原理图如下：



Bagging的本质思想是平均一个多noisy（variance大）但是近似无偏的模型，因此可以减少variance。树是bagging理想的候选者，因为他们能抓住数据中复杂的相互作用结构。在大多数问题中，boosting相对于bagging有压倒性优势，成为更好的选择。

Boosting通过弱学习者的时间演化，成员投一个带权重的票。

Regression: $\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$

因为从bagging中生成的树是id(identically distributed),B颗树的期望与单颗树的期望一样，如果输之间是IID(independent identically distributed),每颗树的variance是 σ^2,则B颗树的variance是 $\frac{\sigma^2}{B}$,假设树之间的关联系数是c,则B颗树的variance是：

$$c\sigma^2 + \frac{1-c}{B}\sigma^2$$

如果c=1，就回到iid情况，如果$c \neq 0$,则上面的第一项不会随着B增大而减小，只有第二项会随着B增大而减小，因此我们要尽量使得树之间的关联系数c趋于0. 这可以通过随机化来实现，对样本和对特征这两方面随机化。

问题：怎么构造具有负关联系数的系统？

Random Forest

[1]. Tree Boosting With XGBoost ↩

贝叶斯方法

逻辑回归与最大熵模型

降维与无监督学习

分类问题与方法

回归问题与方法

K均值EM等聚类算法

正则化方法原理与实践

常用的推荐系统方法与关联规则学习算法

机器学习项目

目录

winequality 可以作为线性回归的练习

从回归到分类，到推进系统，数据集在下面 http://archive.ics.uci.edu/ml/datasets.html 对于回归，可以先用方程产生带噪声的数据，再对这些带噪声的数据进行回归分析。一元高次方程回归。

$y(x) = ax^3 + bx^2 + cx + d + random$ tensorflownews

多元回归分析可以通过求伪逆来求解。

自己需要做的是写一个regression与classfication的类，来实现不同的回归与分类算法，先设计成一个类，如果有必要再设计成一个工厂类。

多元回归分析

多元回归方程

$\hat{y} = \lambda_0 + \lambda_1 x_1 + \lambda_2 x_2 + ... + \lambda_m x_m$

其中b_j是x_j的偏回归系数，$\hat{y}$是样本的估计值。
根据最小值原理，可以求得偏回归系数。

$L(b) = \sum_{i=0}^{n-1}[y_i - (\lambda_0 + \lambda_1 x_{1i} + \lambda_2 x_{2i} + ... + \lambda_m x_{mi})]^2$
变量数为m,样本数为n. 最小二乘法的目的就是找到一组偏回归系数使得损失函数L极小，极端情况就是L=0，则所有样本估计值就是样本的观测值。这只在满秩的情况下存在。一般只是求得L的极小值点。对损失函数求偏导，可以得到：

实际问题可以转化成矩阵分析

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} & 1 \\ x_{21} & x_{22} & \cdots & x_{2m} & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \cdots \\ \lambda_n \\ \lambda_0 \end{bmatrix} = \begin{bmatrix} y_{1o} \\ y_{2o} \\ \cdots \\ y_{no} \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_{2o} \\ \cdots \\ \epsilon_{no} \end{bmatrix}$$

即：
$X * \lambda = Y + \epsilon$

寻找一组λ使得:$L(\lambda) = (X * \lambda - Y)^T (X * \lambda - Y)$极小
利用$L(\lambda)$对向量λ求导可以得到[1]：

$\frac{\partial L(\lambda)}{\partial \lambda} = 2X^T(X\lambda - Y) = 0$

我们也可以得到：$\frac{\partial^2 L(\lambda)}{\partial \lambda^2} = 2X^T X$

求得：$\lambda = (X^T X)^{-1} X^T Y$

$(X^T X)^{-1} X^T$也称为X的伪逆。

多项式拟合

函数的多项式表示方式：$y = \sum_{k=0}^{m} \lambda_k x^k$ 对于n组数据(x_i, y_i)若我们想用多项式去拟合这组数据，就是寻找使下列函数值极小的一组系数λ $L(\lambda) = \frac{1}{2m} \sum_{i=0}^{n-1} (y_i - \sum_{k=0}^{m} \lambda_k x_i^k)^2$ 令：

$\lambda = [\lambda_0, \lambda_1, \cdots, \lambda_m]^T$ $X_i = [x_i^0, x_i^1, \cdots, x_i^m]$ $Y = [Y_0, Y_1, \cdots, Y_{n-1}]^T$

则上面损失函数可以表示为：$L(\lambda) = (X * \lambda - Y)^T (X * \lambda - Y)$

对其求一阶导，结果为0；$\frac{\partial L(\lambda)}{\partial \lambda} = 0$ 最终方程可以化简成如下形式：

$$\begin{bmatrix} \sum_{j=0}^{n-1} x_j^0 & \sum_{j=0}^{n-1} x_j^1 & \cdots & \sum_{j=0}^{n-1} x_j^m \\ \sum_{j=0}^{n-1} x_j^1 & \sum_{j=0}^{n-1} x_j^2 & \cdots & \sum_{j=0}^{n-1} x_j^{m+1} \\ \cdots & \cdots & \cdots & \cdots \\ \sum_{j=0}^{n-1} x_j^m & \sum_{j=0}^{n-1} x_j^{m+1} & \cdots & \sum_{j=0}^{n-1} x_j^{2m} \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \cdots \\ \lambda_m \end{bmatrix} = \begin{bmatrix} \sum_{j=0}^{n-1} y_j * x_j^0 \\ \sum_{j=0}^{n-1} y_j * x_j^1 \\ \cdots \\ \sum_{j=0}^{n-1} y_j * x_j^m \end{bmatrix}$$ 即:$X * \lambda = Y$ 得到

$\lambda = X^{-1}Y$ 若在方程上加上一个正则项，$L(\lambda) = \frac{1}{2m} \sum_{i=0}^{n-1} (y_i - \sum_{k=0}^{m} \lambda_k x_i^k)^2 + \sum_{k=0}^{m} \lambda_k^2$ 得到如下的矩阵：

$$\begin{bmatrix} \sum_{j=0}^{n-1} x_j^0 - 2n & \sum_{j=0}^{n-1} x_j^1 & \cdots & \sum_{j=0}^{n-1} x_j^m \\ \sum_{j=0}^{n-1} x_j^1 & \sum_{j=0}^{n-1} x_j^2 - 2n & \cdots & \sum_{j=0}^{n-1} x_j^{m+1} \\ \cdots & \cdots & \cdots & \cdots \\ \sum_{j=0}^{n-1} x_j^m & \sum_{j=0}^{n-1} x_j^{m+1} & \cdots & \sum_{j=0}^{n-1} x_j^{2m} - 2n \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \cdots \\ \lambda_m \end{bmatrix} = \begin{bmatrix} \sum_{j=0}^{n-1} y_j * x_j^0 \\ \sum_{j=0}^{n-1} y_j * x_j^1 \\ \cdots \\ \sum_{j=0}^{n-1} y_j * x_j^m \end{bmatrix}$$

类似于Ridge回归：但是对于加噪声参数的多项式数据，发现不加正则项能给出正确的结果，加正则项反倒不能。

迭代法求解

求解上面的方程也可以使用迭代法求解：

[1]. 矩阵求导术(上)，https://zhuanlan.zhihu.com/p/24709748 ↩

矩阵微分

参考闲话矩阵求导

向量 y 对标量 x 求导

我们假定所有的向量都是列向量 $\frac{\partial \mathbf{y}}{\partial x}=\begin{bmatrix}\frac{\partial y_1}{\partial x} & \frac{\partial y_2}{\partial x} & \cdots & \frac{\partial y_m}{\partial x}\end{bmatrix}$

标量 y 对向量 x 求导:

$$\frac{\partial y}{\partial \mathbf{x}}=\begin{bmatrix}\frac{\partial y}{\partial x_1}\\ \frac{\partial y}{\partial x_2}\\ \vdots\\ \frac{\partial y}{\partial x_m}\end{bmatrix}$$

向量对向量求导

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}=\begin{bmatrix}\frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1}\\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_2}\\ \vdots & \vdots & \ddots & \vdots\\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n}\end{bmatrix}$$

标量对矩阵求导,

$$\frac{\partial y}{\partial \mathbf{X}}=\begin{bmatrix}\frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{12}} & \cdots & \frac{\partial y}{\partial x_{1q}}\\ \frac{\partial y}{\partial x_{21}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{2q}}\\ \vdots & \vdots & \ddots & \vdots\\ \frac{\partial y}{\partial x_{p1}} & \frac{\partial y}{\partial x_{p2}} & \cdots & \frac{\partial y}{\partial x_{pq}}\end{bmatrix}$$

矩阵对标量求导,

注意有个类似于转置的操作, 因为 Y是mxn矩阵 $\frac{\partial \mathbf{y}}{\partial x}=\begin{bmatrix}\frac{\partial y_{11}}{\partial x} & \frac{\partial y_{21}}{\partial x} & \cdots & \frac{\partial y_{m1}}{\partial x}\\ \frac{\partial y_{12}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{m2}}{\partial x}\\ \vdots & \vdots & \ddots & \vdots\\ \frac{\partial y_{1n}}{\partial x} & \frac{\partial y_{2n}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x}\end{bmatrix}$

用维度分析来解决求导的形式问题

向量对向量的微分 $\frac{\partial(\mathbf{Ax})}{\partial\mathbf{x}}=\begin{bmatrix}\frac{\partial(\mathbf{Ax})_1}{\partial x_1} & \frac{\partial(\mathbf{Ax})_2}{\partial x_1} & \cdots & \frac{\partial(\mathbf{Ax})_m}{\partial x_1}\\ \frac{\partial(\mathbf{Ax})_1}{\partial x_2} & \frac{\partial(\mathbf{Ax})_2}{\partial x_2} & \cdots & \frac{\partial(\mathbf{Ax})_m}{\partial x_2}\\ \vdots & \vdots & \ddots & \vdots\\ \frac{\partial(\mathbf{Ax})_1}{\partial x_n} & \frac{\partial(\mathbf{Ax})_2}{\partial x_n} & \cdots & \frac{\partial(\mathbf{Ax})_m}{\partial x_n}\end{bmatrix}=\begin{bmatrix}a_{11} & a_{21} & \cdots & a_{m1}\\ a_{12} & a_{22} & \cdots & a_{m2}\\ \vdots & \vdots & \ddots & \vdots\\ a_{1n} & a_{2n} & \cdots & a_{mn}\end{bmatrix}=\mathbf{A}^{\mathrm{T}}$ 考虑$\frac{\partial\mathbf{Au}}{\partial\mathbf{x}}$,**A**与x无关，所以**A**肯定可以提出来，只是其形式不知。假如

$\mathbf{A}\in\mathbb{R}^{m\times n},\mathbf{u}\in\mathbb{R}^{n\times 1},\mathbf{x}\in\mathbb{R}^{p\times 1}$ 我们知道最终结果肯定和$\frac{\partial\mathbf{u}}{\partial\mathbf{x}}$有关，注意到$\frac{\partial\mathbf{u}}{\partial\mathbf{x}}\in\mathbb{R}^{p\times n}$,于是**A**只能转置以后添在后面，因此：$\frac{\partial\mathbf{Au}}{\partial\mathbf{x}}=\frac{\partial\mathbf{u}}{\partial\mathbf{x}}\mathbf{A}^{\mathrm{T}}$

$dL=\frac{\partial\mathbf{L}^{\mathrm{T}}}{\partial\mathbf{w}}d\mathbf{w}$

再考虑如下问题：$\frac{\partial\mathbf{x}^{\mathrm{T}}\mathbf{Ay}}{\partial\mathbf{x}},\mathbf{x}\in\mathbb{R}^{m\times 1},\mathbf{y}\in\mathbb{R}^{n\times 1}$ 其中A与x无关，我们知道 $\frac{\partial\mathbf{x}^{\mathrm{T}}\mathbf{Ay}}{\partial\mathbf{x}}\in\mathbb{R}^{m\times 1}$ 因此 $\frac{\partial\mathbf{x}^{\mathrm{T}}\mathbf{Ay}}{\partial\mathbf{x}}=f(A,y)+g(\frac{\partial\mathbf{y}}{\partial\mathbf{x}},x^TA)$ $\mathbf{A}\in\mathbf{R}^{m\times n},\mathbf{y}\in\mathbf{R}^{n\times 1},\mathbf{x^TA}\in\mathbf{R}^{1\times n},\frac{\partial\mathbf{y}}{\partial\mathbf{x}}\in\mathbf{R}^{m\times n}$ 因此，为了满足 $\frac{\partial\mathbf{x}^{\mathrm{T}}\mathbf{Ay}}{\partial\mathbf{x}}\in\mathbb{R}^{m\times 1}$，我们可以知道函数f,g的形式。最终有：$\frac{\partial(\mathbf{x}^{\mathrm{T}}\mathbf{A})\mathbf{y}}{\partial\mathbf{x}}=\frac{\partial\mathbf{y}}{\partial\mathbf{x}}\mathbf{A}^{\mathrm{T}}\mathbf{x}+\mathbf{Ay}$ 当$\mathbf{x}=\mathbf{y}$时，$\frac{\partial(\mathbf{x}^{\mathrm{T}}\mathbf{A})\mathbf{y}}{\partial\mathbf{x}}=(\mathbf{A}^{\mathrm{T}}+\mathbf{A})\mathbf{x}$

最后一个例子(还没解出来)：$\frac{\partial\mathbf{a}^T\mathbf{x}\mathbf{x}^T\mathbf{b}}{\partial\mathbf{x}},\mathbf{a},\mathbf{b},\mathbf{x}\in\mathbb{R}^{m\times 1}$ 知道$\frac{\partial\mathbf{a}^T\mathbf{x}\mathbf{x}^T\mathbf{b}}{\partial\mathbf{x}}\in\mathbb{R}^{m\times 1}$ 还需要对迹形式进行求解.

正则化

L0 is OMP-k

Ensemble learning（团体学习）

利用多个学习方法来获得更好的预测能力，机器学习中的系综学习的样本是有限的，但是统计力学中的系综方法其样本是无限的。机器学习中的系综方法包括Bagging,boosting

Bagging算法（Bootstrap aggregating）

参考 是一种团体学习方法,可以看成事一种圆桌会议，或者投票选举的形式，其思想是“群众的眼光是雪亮的”，可以训练多个模型，之后将这些模型进行加权组合，一般这类方法的效果，都会好于单个模型的效果。在实践中，在特征一定的情况下，大家总是使用Bagging的思想去提升效果。算法步骤 给定一个大小为n的训练集D，Bagging算法从中均匀、有放回地（即使用自助抽样法）选出m个大小为n'的子集D_i，作为新的训练集。在这m个训练集上使用分类、回归等算法，则可得到m个模型，同一个训练集中的成员可以有重重复，再通过取平均值、取多数票等方法，即可得到Bagging的结果。

Boosting

在Bagging方法中，我们假设每个训练样本的权重都是一致的；而Boosting算法则更加关注错分的样本，越是容易错分的样本，约要花更多精力去关注。对应到数据中，就是该数据对模型的权重越大，后续的模型就越要拼命将这些经常分错的样本分正确。最后训练出来的模型也有不同权重，所以boosting更像是会整，级别高，权威的医师的话语权就重些。训练：先初始化每个训练样本的权重相等为1/d，d为样本数量；之后每次使用一部分训练样本去训练弱分类器，且只保留错误率小于0.5的弱分类器，对于分对的训练样本，将其权重 调整为 error(Mi)/(1-error(Mi))，其中error(Mi)为第i个弱分类器的错误率（降低正确分类的样本的权重，相当于增加分错样本的权重）；

测试：每个弱分类器均给出自己的预测结果，且弱分类器的权重为log(1-error(Mi))/error(Mi)) 权重最高的类别，即为最终预测结果。

在adaboost中，弱分类器的个数的设计可以有多种方式，例如最简单的就是使用一维特征的树作为弱分类器。

adaboost在一定弱分类器数量控制下，速度较快，且效果还不错。

我们在实际应用中使用adaboost对输入关键词和推荐候选关键词进行相关性判断。随着新的模型方法的出现，adaboost效果已经稍显逊色，我们在同一数据集下，实验了GBDT和adaboost，在保证召回基本不变的情况下，简单调参后的Random Forest准确率居然比adaboost高5个点以上，效果令人吃惊。。。。

Bagging和Boosting都可以视为比较传统的集成学习思路。现在常用的Random Forest，GBDT，GBRank其实都是更加精细化，效果更好的方法。后续会有更加详细的内容专门介绍。

训练神经网络的经验

第一步(一个月)，公式推导必须会，核心思想会，并且要融会贯通。

1. SVM
2. LR, LTR
3. 决策树，RF，GBDT，xgboost
4. 贝叶斯
5. 降维：PCA, SVD
6. BP的公式推导，以及写一个简单的神经网络，并跑个小的数据。用python,numpy。
7. LeetCode上面刷Easy与Medium的题
8. 看面经，找面试常遇到的问题

第二步是最优化总结(半个月)，总结常用的优化算法，比如梯度下降，牛顿，拟牛顿，trust region，二次规划。

第六章 C++

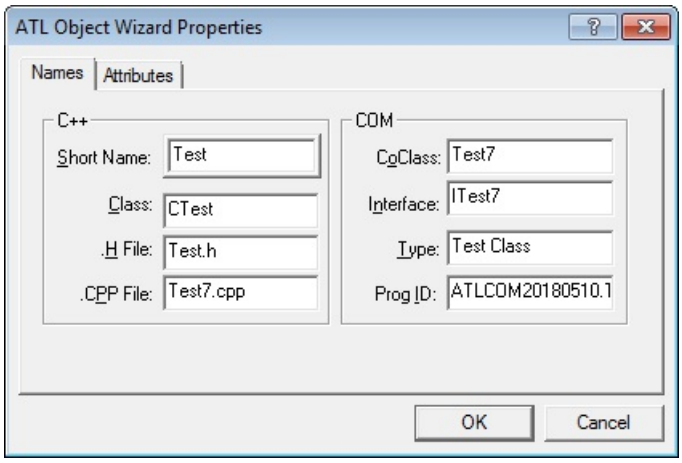
第一节 基本语法

接口:即抽象类,就是包含至少一个纯虚函数的类

```
一个类里面实现多种接口Iinterface, IinterfaceB, IinterfaceC
    IE84TPTimeOutUIAck : public IUnknown
    {
    public:
        virtual HRESULT STDMETHODCALLTYPE ResponseTpTimeOutUIAck(
            /* [in] */ short nPortNo,
            /* [in] */ short nTpNo) = 0;

    };
```

New ATL object 时,选择simple object,然后属性设置如下:则可以实现一个类中有多组接口函数。



C++通过ATL来实现接口的继承。

```
// Cr3halObj
class ATL_NO_VTABLE Cr3halObj :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<Cr3halObj, &CLSID_r3halObj>,
    public IConnectionPointContainerImpl<Cr3halObj>,
    public Ir3halObj,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IHALEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE84Events>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IMMIEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE90Events>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE116Events>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IFFUEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IWaferProtrusionEvent>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_ISignalTowerDevice>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE84HOAVBLEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE84TPTimeOutUIAck>, //TP3 connection point
    public IR3HALDiag
{

DECLARE_EMUCLASSFACTORY_SINGLETON(Cr3halObj)
```

COM组件接口继承的实现 https://blog.csdn.net/dingbaosheng/article/details/624504

TL学习笔记03

继承

C++:

C#:

http://www.cnblogs.com/flyinthesky/archive/2008/06/18/1224774.html

不能初始化的类被叫做抽象类，它们只提供部分实现，但是另一个类可以继承它并且能创建它们的实例。"一个包含一个或多个纯虚函数的类叫抽象类，抽象类不能被实例化，进一步一个抽象类只能通过接口和作为其它类的基类使用.

"抽象类能够被用于类，方法，属性，索引器和事件，使用abstract 在一个类声明中表示该类倾向要作为其它类的基类

成员被标示成abstract，或被包含进一个抽象类，必须被其派生类实现.

一个抽象类可以包含抽象和非抽象方法，当一个类继承于抽象类，那么这个派生类必须实现所有的的基类抽象方法。

一个抽象方法是一个没有方法体的方法。

1. virtual修饰的方法必须有实现(哪怕是仅仅添加一对大括号),而abstract修饰的方法一定不能实现。
2. virtual可以被子类重写，而abstract必须被子类重写，
3. 如果类成员被abstract修饰，则该类前必须添加abstract，因为只有抽象类才可以有抽象方法。
4. 无法创建abstract类的实例，只能被继承无法实例化，
5. C#中如果要在子类中重写方法，必须在父类方法前加virtual，在子类方法前添加override,这样就避免了程序员在子类中不小心重写了父类方法。
6. abstract方法必须重写，virtual方法必须有实现(即便它是在abstract类中定义的方法).

mRet = TestHalfWafer(token, ref nSeq, true); //C# ref

模板类与模板函数

Why can templates only be implemented in the header file?

Clarification: header files are not the _only _portable solution. But they are the most convenient portable solution Error LNK2019 unresolved external symbol "public: int __thiscall Algo<int>::LongestIncreaseSubsequence(int * const,int)" (?LongestIncreaseSubsequence@?$Algo@H@@QAEHQAHH@Z) referenced in function _main Algo C:\Data\ShareFolder\Group\Tim\Code\Algo\Algo\Main.obj

当模板声明在头文件，实现在cpp中，一般会出现上面的问题，原因是
Algo<int> Alg实例化的时候，编译器会去创建一个int型的Algo的新类，以及LongestIncreaseSubsequence(T arr[], int arrSize)方法，因此编译器会去找这个方法的实现，如果没有找到，自然会报错。因此有如下两个方法

1. 在头文件中实现
2. 在头文件中声明，再在cpp中显示实例化(explicit instantiations) template class Algo<int>;并实现模板函数

方式1:实现在头文件里

```
#pragma once
#include <algorithm>
using namespace std;

template<class T>
class Algo
{
public:
    Algo() {};
    ~Algo() {};
public:
    int LongestIncreaseSubsequence(T arr[], int arrSize)
    {
        int *maxLen = new int[arrSize]();
        for (int i = 0; i < arrSize; i++)
        {
            maxLen[i] = 1;
        }
        for (int j = 1; j < arrSize; j++)
        {
            int maxLenJ = 1;
            for (int i = 0; i < j; i++)
            {
                if (arr[i] < arr[j])
                {
```

```
                    maxLenJ = maxLen[i] + 1;
                    maxLen[j] = std::max({ maxLen[j] ,maxLenJ });
                }
            }
        }
        return maxLen[arrSize - 1];
    }
};
```

```
//调用函数
#include "stdafx.h"
#include "Algo.h"
#include <iostream>
using namespace std;

int main()
{
    int param[6] = {5,3,4,8,6,7};
    int arrSize = sizeof(param) / sizeof(int);
    Algo<int> Alg;
    int maxLength = Alg.LongestIncreaseSubsequence(param, arrSize);
    cout << "Max length: " << maxLength << endl;
    return 0;
}
```

方式二:在头文件中声明，CPP中定义并对每种实例进行声明

```
Algo.h
#pragma once
#include <algorithm>
using namespace std;

template<class T>
class Algo
{
public:
    Algo() {};
    ~Algo() {};
public:
    int LongestIncreaseSubsequence(T arr[], int arrSize);
}
```

```
// Algo.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "Algo.h"
#include <algorithm>
using namespace std;
```

```
template class Algo<int>; //必须先声明，才可以用
template class Algo<double>;
template class Algo<char>;
template<class T>
int Algo<T>::LongestIncreaseSubsequence(T arr[], int arrSize)
{
    int *maxLen = new int[arrSize]();
    for (int i = 0; i < arrSize; i++)
    {
        maxLen[i] = 1;
    }
    for (int j = 1; j < arrSize; j++)
    {
        int maxLenJ = 1;
        for (int i = 0; i < j; i++)
        {
            if (arr[i] < arr[j])
            {
                maxLenJ = maxLen[i] + 1;
                maxLen[j] = std::max({ maxLen[j] ,maxLenJ });
            }
        }
    }
    return maxLen[arrSize - 1];
}
```

```
If all you want to know is how to fix this situation, read the next two FAQs.
But in order to understand why things are the way they are, first accept these facts:

A template is not a class or a function. A template is a “pattern” that the compiler uses to
generate a family of classes or functions.
In order for the compiler to generate the code, it must see both the template definition (not
 just declaration) and the specific types/whatever used to “fill in” the template. For exampl
e, if you’re trying to use a Foo<int>, the compiler must see both the Foo template and the fa
ct that you’re trying to make a specific Foo<int>.
Your compiler probably doesn’t remember the details of one .cpp file while it is compiling an
other .cpp file. It could, but most do not and if you are reading this FAQ, it almost definit
ely does not. BTW this is called the “separate compilation model.”
Now based on those facts, here’s an example that shows why things are the way they are. Suppo
se you have a template Foo defined like this:

template<typename T>
class Foo {
public:
  Foo();
  void someMethod(T x);
private:
  T x;
};
Along with similar definitions for the member functions:
```

```
template<typename T>
Foo<T>::Foo()
{
  // ...
}
template<typename T>
void Foo<T>::someMethod(T x)
{
  // ...
}
Now suppose you have some code in file Bar.cpp that uses Foo<int>:

// Bar.cpp
void blah_blah_blah()
{
  // ...
  Foo<int> f;
  f.someMethod(5);
  // ...
}
Clearly somebody somewhere is going to have to use the “pattern” for the constructor definiti
on and for the someMethod() definition and instantiate those when T is actually int.
But if you had put the definition of the constructor and someMethod() into file Foo.cpp, the
compiler would see the template code when it compiled Foo.cpp and it would see Foo<int> when
it compiled Bar.cpp,
but there would never be a time when it saw both the template code and Foo<int>. So by rule #
2 above, it co
```

```
uld never generate the code for  Foo<int>::someMethod().
A note to the experts: I have obviously made several simplifications above. This was intentio
nal so please don’t complain too loudly. If you know the difference between a .cpp file and a
 compilation unit, the difference between a class template and a template class, and the fact
 that templates really aren’t just glorified macros, then don’t complain: this particular que
stion/answer wasn’t aimed at you to begin with. I simplified things so newbies would “get it,
” even if doing so offends some experts.
```

const对象只能访问**const**成员函数。因为**const**对象表示其不可改变，而非**const**成员函数可能在内部改变了对象，所以不能调用。

而非**const**对象既能访问**const**成员函数，也能访问非**const**成员函数，因为非**const**对象表示其可以改变。

```
#ifndef _MATRIX_H
#define _MATRIX_H
#include <iostream>
#include <algorithm>
using namespace std;
```

```
template<class T>
class CMatrix
{
public:
    CMatrix() {};
    CMatrix(int rows, int columns);
    CMatrix(const CMatrix&);
    ~CMatrix() {};
public:
    void Set(int row, int column, T val);
    T Get(int row, int column) const;
    CMatrix operator +(const CMatrix &mat2);
    CMatrix operator -(const CMatrix &mat2);
    CMatrix operator *(const CMatrix &mat2);
    CMatrix operator *(const T div);
    CMatrix operator /(const T div);
    CMatrix I(int n);
    int Rows() const{ return mRows; }
    int Columns() const{ return mColumns; }
    int Length() { return mRows*mColumns; }
    friend ostream &operator<<(ostream &os, const CMatrix &mat)
    {
        int row = mat.Rows();
        int col = mat.Columns();
        //mat是const，因此只能访问const成员函数

        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < col; j++)
            {
                os << fixed << mat.Get(i, j) << '\t';
            }
            if (i < row)
            {
                os << "\n";
            }
        }
        os << "\n";
        return os;
    };

private:
    int mRows;
    int mColumns;
    T* startElement;

};
#endif
```

COM连接点

COM连接点 - 最简单的例子

```
AtlAdvise(spCar, sinkptr, __uuidof(_IMyCarEvents), &cookies);
sinkptr指向了一个CComObject<CSink>
CComObject<CSink>* sinkptr = nullptr;
CComObject<CSink>::CreateInstance(&sinkptr);
```

换句话说，一旦将sink对象成功挂载到了COM对象，那么sink对象的生命周期就由对应的COM(spCar)对象来管理。

一旦挂载成功，sink对象就托管给对应的COM对象了，如果对应的COM对象析构了，那么所有它管理的sink对象也就释放了。

```
CLR是什么？
```

COM Interop(互操作)

引言

- 平台调用
- **C++ Interop(互操作)**
- **COM Interop(互操作)**

一、引言

这个系列是在C#基础知识中遗留下来的一个系列的，因为在C# 4.0中的一个新特性就是对COM互操作改进，然而COM互操作性却是.NET平台下其中一种互操作技术，为了帮助大家更好的了解.NET平台下的互操作技术，所以才有了这个系列。然而有些朋友们可能会有这样的疑问——“为什么我们需要掌握互操作技术的呢？”对于这个问题的解释就是——掌握了.NET平台下的互操作性技术可以帮助我们在.NET中调用非托管的dll和COM组件。.NET是建立在操作系统的之上的一个开发框架，其中.NET 类库中的类也是对Windows API的抽象封装，然而.NET类库不可能对所有Windows API进行封装，当.NET中没有实现某个功能的类，然而该功能在Windows API被实现了，此时我们完全没必要去自己在.NET中自定义个类，这时候就可以调用Windows API 中的函数来实现，此时就涉及到托管代码与非托管代码的交互，此时就需要使用到互操作性的技术来实现托管代码和非托管代码更好的交互。.NET 平台下提供了3种互操作性的技术：

1. Platform Invoke(P/Invoke)，即平台调用,主要用于调用C库函数和Windows API
2. C++ Introp，主要用于Managed C++(托管C++)中调用C++类库

3. COM Interop, 主要用于在.NET中调用COM组件和在COM中使用.NET程序集。

下面就对这3种技术分别介绍下。

二、平台调用

使用平台调用的技术可以在托管代码中调用动态链接库(Dll)中实现的非托管函数，如Win32 Dll和C/C++ 创建的dll。看到这里，有些朋友们应该会有疑问——在怎样的场合我们可以使用平台调用技术来调用动态链接库中的非托管函数呢？

这个问题就如前面引言中说讲到的一样，当在开发过程中，.NET类库中没有提供相关API然而Win32 API 中提供了相关的函数实现时，此时就可以考虑使用平台调用的技术在.NET开发的应用程序中调用Win32 API中的函数；

然而还有一个使用场景就是——由于托管代码的效率不如非托管代码，为了提高效率，此时也可以考虑托管代码中调用C库函数。

2.1 在托管代码中通过平台调用来调用非托管代码的步骤

(1). 获得非托管函数的信息，即dll的名称，需要调用的非托管函数名等信息

(2). 在托管代码中对非托管函数进行声明，并且附加平台调用所需要属性

(3). 在托管代码中直接调用第二步中声明的托管函数

2.2 平台调用的调用过程

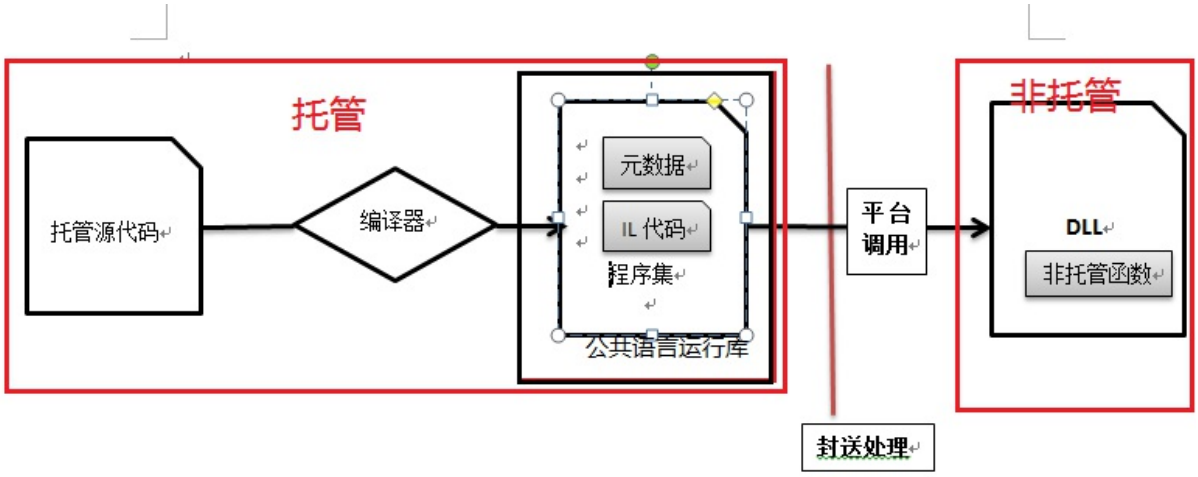
(1) 查找包含该函数的DLL，当需要调用某个函数时，当然第一步就需要知道包含该函数的DLL的位置，所以平台调用的第一步也就是查找DLL，其实在托管代码中调用非托管代码的调用过程可以想象成叫某个人做事情，首先我们要找到那个人在哪里(即查找函数的**DLL**过程)，找到那个人之后需要把要做的事情告诉他(相当于加载**DLL**到内存中和传入参数)，最后让他去完成需要完成的事情(相当于让非托管函数去执行任务)。

(2) 将找到的DLL加载到内存中。

(3) 查找函数在内存中的地址并把其参数推入堆栈，来封送所需的数据。CLR只会在第一次调用函数时，才会去查找和加载DLL，并查找函数在内存中的地址。当函数被调用过一次之后，CLR会将函数的地址缓存起来，CLR这种机制可以提高平台调用的效率。在应用程序域被卸载之前，找到的DLL都一直存在于内存中。

(4) 执行非托管函数。

平台调用的过程可以通过下图更好地理解：



三、C++ Interop

第二部分主要向大家介绍了第一种互操作性技术，然后我们也可以使用C++ Interop技术来实现与非托管代码进行交互。然而C++ Interop 方式有一个与平台调用不一样的地方，就是C++ Interop 允许托管代码和非托管代码存在于一个程序集中，甚至同一个文件中。C++ Interop 是在源代码上直接链接和编译非托管代码来实现与非托管代码进行互操作的，而平台调用是加载编译后生成的非托管DLL并查找函数的入口地址来实现与非托管函数进行互操作的。**C++ Interop**使用托管**C++**来包装非托管**C++**代码，然后编译生成程序集，然后再托管代码中引用该程序集，从而来实现与非托管代码的互操作。关于具体的使用和与平台调用的比较，这里就不多介绍，我将会在后面的专题中具体介绍。

COM(Component Object Model. 组件对象模型)是微软之前推荐的一种开发技术，由于微软过去十多年里开发了大量的COM组件，

然而不可能再使用.Net技术重写这些COM组件实现的功能，所以为了解决在.Net中的托管代码能够调用COM组件中问题，.NET平台下提供了COM Interop，即COM互操作技术，COM interop不仅支持在托管代码中使用COM组件，而且支持向COM组件中使用.NET程序集。

1.在.NET中使用COM组件

在.NET中使用COM对象，主要有三种方法：

1. 使用TlbImpl工具为COM组件创建一个互操作程序集来绑定早期的COM对象，这样就可以在程序中添加互操作程序集来调用COM对象
2. 通过反射来后期绑定COM对象
3. 通过P/Invoke创建COM对象或使用C++ Interop为COM对象编写包装类

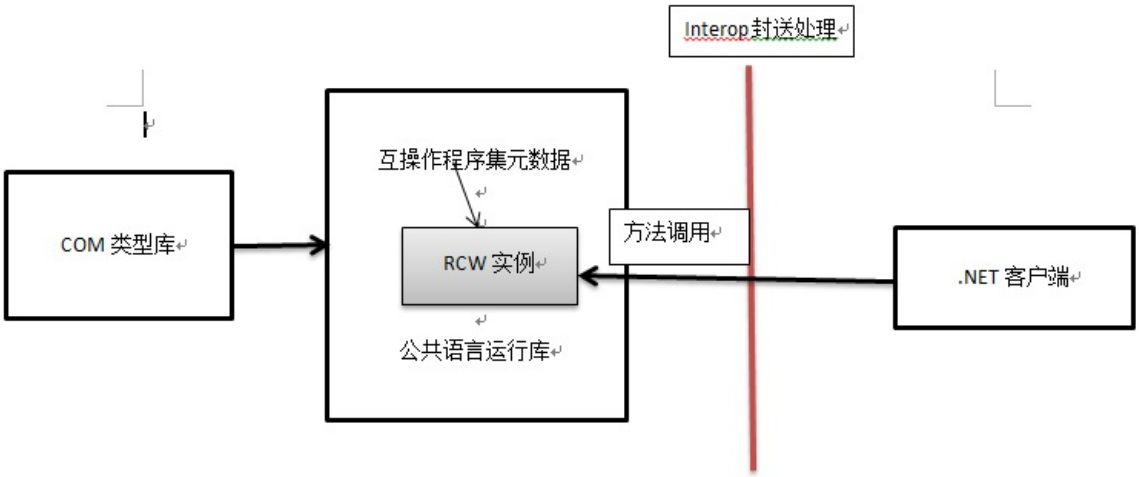
但是我们经常使用的都是方法一，下面介绍下使用方法一在.NET 中使用COM对象的步骤：

1. 找到要使用的COM 组件并注册它。使用 regsvr32.exe 注册或注销 COM DLL。
2. 在项目中添加对 COM 组件或类型库的引用。

添加引用时，**Visual Studio** 会用到**TIbimp.exe**（类型库导入程序），**TIbimp.exe**程序将生成一个 **.NET Framework** 互操作程序集。该程序集又称为运行时可调用包装 **(RCW)**，其中包含了包装**COM**组件中的类和接口。**Visual Studio** 将生成组件的引用添加至项目。

1. 创建RCW中类的实例，这样就可以使用托管对象一样来使用COM对象。

下面通过一个图更好地说明在.NET中使用COM组件的过程：

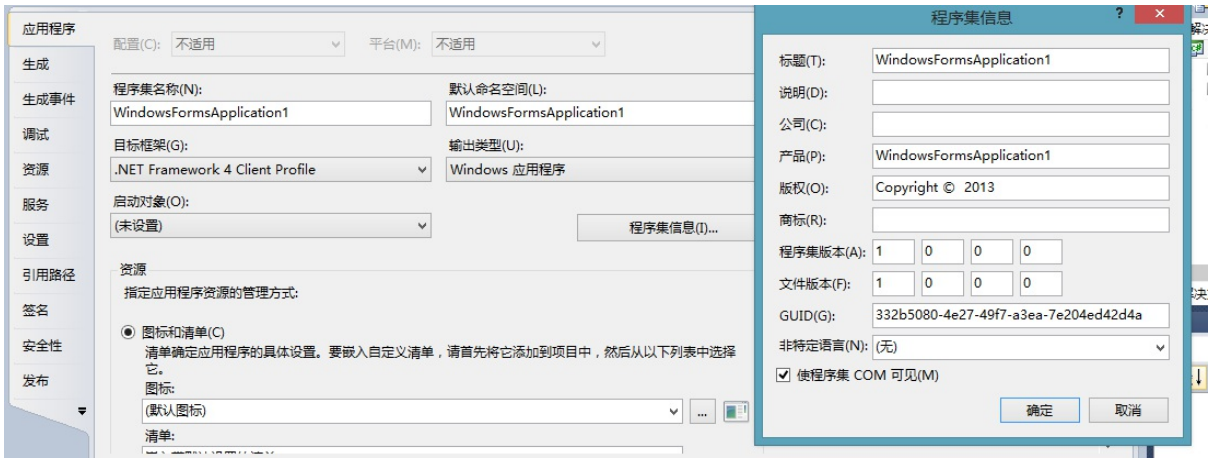


在COM中使用.NET程序集

.NET 公共语言运行时通过COM可调用包装（COM Callable Wrapper,即CCW）来完成与COM类型库的交互。CCW可以使COM客户端认为是在与普通的COM类型交互，同时使.NET组件认为它正在与托管应用程序交互。在这里CCW是非托管COM客户端与托管对象之间的一个代理。CCW既可以维护托管对象的生命周期，也负责数据类型在COM和.NET之间的相互转换。实现在COM使用.NET 类型的基本步骤如：

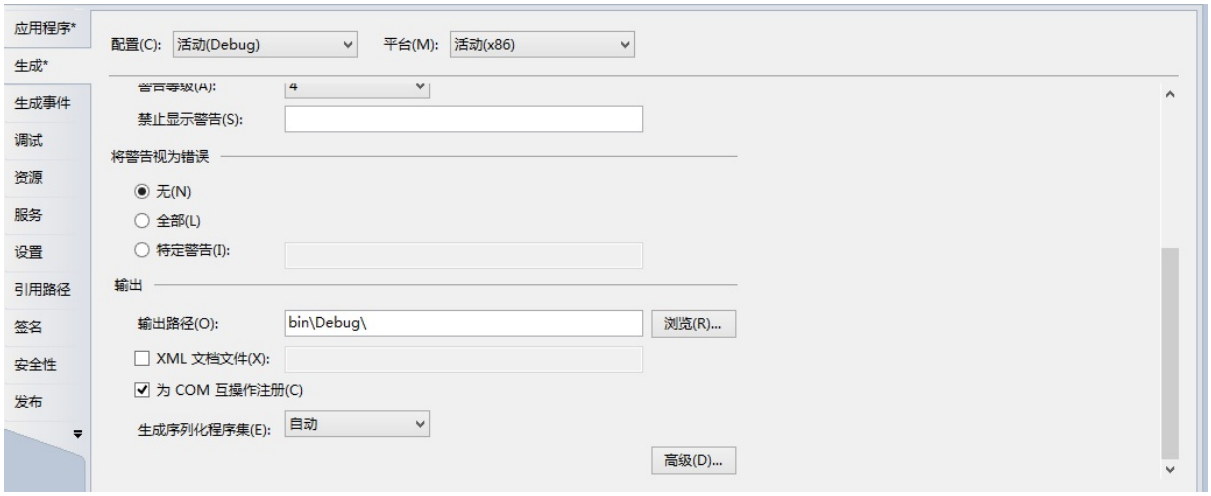
1. 在C#项目中添加互操作特性

   可以修改C#项目属性使程序集对COM可见。右键解决方案选择属性，在“应用程序标签”中选择“程序集信息”按钮，在弹出的对话框中选择 “使程序集COM可见” 选项，如下图所示：



1. 生成COM类型库并对它进行注册以供COM客户端使用

在“生成”标签中，选中 “为COM互操作注册”选项，如下图：



勾选“为COM互操作注册”选项后，Visual Studio会调用类型库导出工具(Tlbexp.exe)为.NET程序集生成COM类型库再使用程序集注册工具(Regasm.exe)来完成对.NET程序集和生成的COM类型库进行注册，这样COM客户端可以使用CCW服务来对.NET对象进行调用了。

总结

介绍到这里，本专题的内容就结束，本专题主要对.NET 提供的互操作的技术做了一个总的概括，在后面的专题中将会对具体的技术进行详细的介绍和给出一些简单的使用例子。

驱动

```
WINBASEAPI
BOOL
WINAPI
DeviceIoControl(
    HANDLE hDevice,
    DWORD dwIoControlCode,
    LPVOID lpInBuffer,
    DWORD nInBufferSize,
    LPVOID lpOutBuffer,
    DWORD nOutBufferSize,
    LPDWORD lpBytesReturned,
    LPOVERLAPPED lpOverlapped
    );
```

是WinAPI，负责与硬件打交道，收发数据，然后在我们的驱动程序DispatchControl函数中，去解析DeviceIoControl传送的数据

```
// Control.cpp -- IOCTL handlers for usb42 driver
// Copyright (C) 1999 by Walter Oney
// All rights reserved

#include "stddcls.h"
#include "driver.h"
#include "ioctls.h"

///////////////////////////////////////////////////////////////////////////////

#pragma PAGEDCODE

NTSTATUS DispatchControl(PDEVICE_OBJECT fdo, PIRP Irp)
    {                           // DispatchControl
    PAGED_CODE();
    PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

    NTSTATUS status = IoAcquireRemoveLock(&pdx->RemoveLock, Irp);
    if (!NT_SUCCESS(status))
        return CompleteRequest(Irp, status, 0);
    ULONG info = 0;

    PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);
    ULONG cbin = stack->Parameters.DeviceIoControl.InputBufferLength;
    ULONG cbout = stack->Parameters.DeviceIoControl.OutputBufferLength;
    ULONG code = stack->Parameters.DeviceIoControl.IoControlCode;

    switch (code)
        {                       // process request
```

```
    case IOCTL_USB42_READ:
        {                       // IOCTL_USB42_READ
        if (cbout != 64)
            {
            status = STATUS_INVALID_PARAMETER;
            break;
            }

        URB urb;
        UsbBuildInterruptOrBulkTransferRequest(&urb, sizeof(_URB_BULK_OR_INTERRUPT_TRANSFER),

            pdx->hpipe, Irp->AssociatedIrp.SystemBuffer, NULL, cbout, USBD_TRANSFER_DIRECTION
_IN | USBD_SHORT_TRANSFER_OK, NULL);
        status = SendAwaitUrb(fdo, &urb);
        if (!NT_SUCCESS(status))
            KdPrint(("USB42 - Error %X (USBD status code %X) trying to read endpoint\n", stat
us, urb.UrbHeader.Status));
        else
            info = cbout;
        break;
        }                       // IOCTL_USB42_READ

    default:
        status = STATUS_INVALID_DEVICE_REQUEST;
        break;

        }                       // process request

    IoReleaseRemoveLock(&pdx->RemoveLock, Irp);
    return CompleteRequest(Irp, status, info);
    }                           // DispatchControl
```

结束