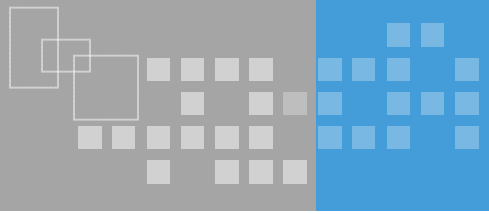
NLPR, CASIA

教你认识SVM和使用开源工具LibSVM

夏睿　rxia@nlpr.ia.ac.cn

提纲



- SVM能解决什么问题？
  - 从机器学习谈起
  - 分类与回归
- 为什么选择SVM？
  - 从ANN到SVM
  - SVM的两个核心思想
  - SVM算法
- 怎么用SVM？
  - 用LibSVM做分类
  - 用LibSVM做回归
  - Python平台下的LibSVM

从机器学习谈起

- 目的

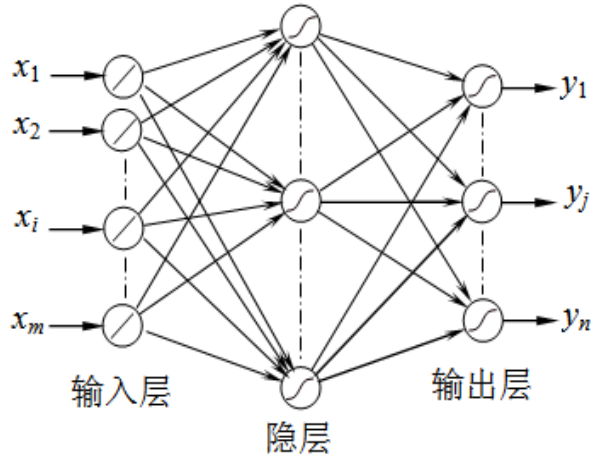
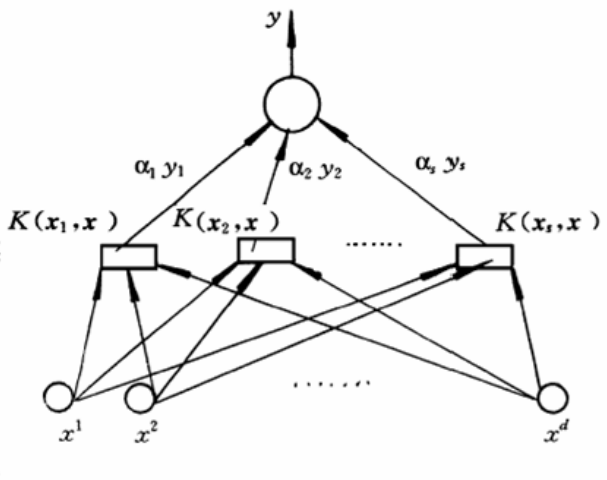
  根据给定的训练样本，对某系统输入输出之间依赖关系的估计，使它能够对未知输出作出尽可能准确的预测。

- 机器学习的三个基本问题
  - 模式识别——分类
  - 函数拟合——回归
  - 概率密度估计

- 数学表述
  - 给定条件：n个独立同分布观测样本 $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots (\mathbf{x}_n, y_n)$
  - 目标：求一个最优函数 $f(\mathbf{x}, \boldsymbol{\omega}^*)$
  - 最理想的要求：最小化期望风险 $R(\boldsymbol{\omega}) = \int L(y, f(\mathbf{x}, \boldsymbol{\omega}))dF(\mathbf{x}, y)$

- 机器学习方法的代表
  - 人工神经网络（ANN）& 支持向量机（SVM）

ANN VS SVM

ANN	VS	SVM
	→	
传统统计学	→	统计学习理论
经验风险最小化(ERM)	→	结构风险最小化(SRM)
过学习	→	推广能力的控制 （过人之处1）
训练较慢	→	训练较快
容易陷入局部最小	→	全局最优解
完全的黑匣子，高度非线性，人工难以理解和干预	→	这个黑匣子要透明一点，怎么个透明法？ （过人之处2）

广义最优分类面

- SVM的两点过人之处
  - 推广能力的控制
    ——广义最优分类面
  - 处理非线性问题的方式
    ——核函数

- SVM核心思想（一）：广义最优分类面
  - 最优分类面（线）方程 $y = \boldsymbol{\omega} \cdot \mathbf{x} + b$
  - 最大的分类间隔 $\text{margin} = 2 / \|\boldsymbol{\omega}\|$
    使分类间隔最大实际上就是对推广能力的控制
  - SVM得名由来：支持向量是什么？
  - 怎么求解广义最有分类面？—— $\boldsymbol{\omega}^*$
    这是一个不等式约束下的二次函数寻优问题：
    利用Lagrange优化方法将原问题转化为其对偶问题。

求解广义最优分类面 - 1

- 线性可分下的最优分类面求解

$$\min \frac{1}{2}\|\boldsymbol{\omega}\|^2$$

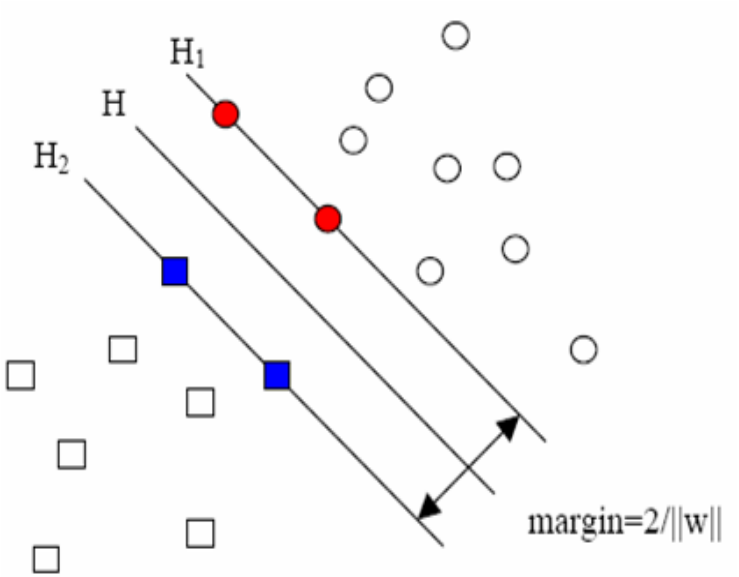
$$s.t. \quad y_i(\boldsymbol{\omega}\cdot\mathbf{x}_i+b)\geq 1(i=1,2,\ldots,n)$$

原始问题

$$\min \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j)-\sum_{i=1}^{n}\alpha_i$$

对偶问题

$$s.t. \sum_{i=1}^{n} y_i\alpha_i=0, \alpha_i\geq 0, i=1,\ldots,n$$



- 最优解需满足的条件

$$\alpha_i(y_i(\boldsymbol{\omega}\cdot\mathbf{x}_i+b)-1)=0, i=1,\ldots,n$$

- 支持向量

$\{\alpha_i, i=1,\ldots,n; \alpha_i>0\}$

对应于图中的红点和蓝点 $y_i(\boldsymbol{\omega}\cdot\mathbf{x}_i+b)=1$

求解广义最优分类面 - 2

- 最优分类面函数

$$\begin{cases} \boldsymbol{\omega}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i \\ b^* = \frac{1}{2}((\boldsymbol{\omega}^*)^{\mathrm{T}} \mathbf{x}_+ + (\boldsymbol{\omega}^*)^{\mathrm{T}} \mathbf{x}_-) \end{cases}$$

$$f(\mathbf{x}) = \mathrm{sgn}\{\boldsymbol{\omega}^* \cdot \mathbf{x} + b^*\} = \mathrm{sgn}\{\sum_{i=1}^{n} \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}) + b^*\}$$

- 松弛项，折衷考虑最小错分样本和最大分类间隔

$$\min(\frac{1}{2}\|\boldsymbol{\omega}\|^2 + C\sum_{i=1}^{n} \xi_i)$$

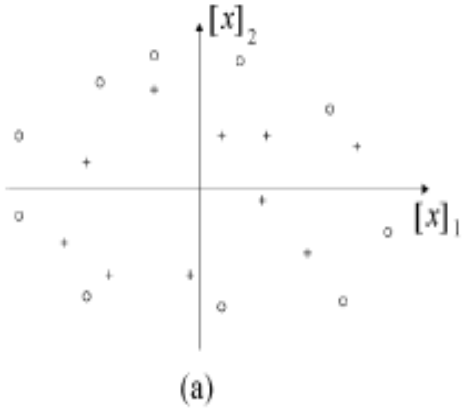
$$s.t. y_i(\boldsymbol{\omega} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i (i = 1, 2, ..., n), \xi_i \geq 0$$

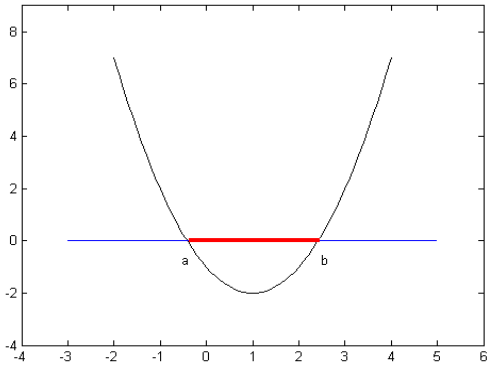
- 非线性可分怎么办？

非线性的解决

- 解决非线性问题

  通过合理的非线性变化，将非线性问题转化为某个高维空间的线性问题。





- SVM怎么实现低维到高维的变换
  - 从广义最优分类面能得到什么启示？

$$\mathbf{h}=\phi(\mathbf{x}),\ g(\mathbf{x})=f(\mathbf{h})=\mathrm{sgn}\{\boldsymbol{\omega}^*\cdot\mathbf{h}+b^*\}=\mathrm{sgn}\{\sum_{i=1}^{n}\alpha_i^* y_i(\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}))+b^*\}$$

  - 关键问题：怎么计算 $\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x})$

    需要知道变换的具体形式吗？灾难维数？NO!

核函数与SVM算法

- SVM核心思想（二）：核函数 $(\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j))=K(\mathbf{x}_i,\mathbf{x}_j)=\varphi(\mathbf{x}_i\cdot\mathbf{x}_j)$
  - 低维内积的函数
  - 对应某一高维变换空间的内积
  - 常用的核函数

Linear	Polynomial	RBF	sigmoid
$K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^T\mathbf{x}_j$	$(\gamma\mathbf{x}_i^T\mathbf{x}_j+r)^d$	$\exp(-\gamma\left\|\mathbf{x}_i-\mathbf{x}_j\right\|^2)$	$\tanh(\gamma\mathbf{x}_i^T\mathbf{x}_j+r)$

- 非线性问题的SVM算法

$$\min_{\alpha}\ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}y_iy_j\alpha_i\alpha_jK\left(\mathbf{x}_i,\mathbf{x}_j\right)-\sum_{j=1}^{n}\alpha_j$$

$$s.t.\quad \sum_{i=1}^{n}y_i\alpha_i=0$$

$$0\le\alpha_i\le C,i=1,\cdots l$$

$$f(x)=\mathrm{sgn}(\sum_{i=1}^{l}y_i\alpha_iK(\mathbf{x}_i,\mathbf{x}))+b^*),\ b^*=y_j-\sum_{i=1}^{l}y_i\alpha_iK(\mathbf{x}_i,\mathbf{x}_j)\quad j\in\{j\mid 0<\alpha_j^*<C\}$$

SVM开源工具——LibSVM

- 为什么选择LibSVM (http://*www.csie.ntu.edu.tw/~cjlin/libsvm*)
  - Different SVM formulations
  - Efficient multi-class classification
  - Cross validation for model selection
  - Probability estimates
  - Weighted SVM for unbalanced data
  - Both C++ and Java sources
  - GUI demonstrating SVM classification and regression
  - Python, R (also Splus), MATLAB, Perl, Ruby, Weka, Common LISP and LabVIEW interfaces. C# .NET code is available.
  - Automatic model selection which can generate contour of cross valiation accuracy.

- 使用LibSVM的一些准备工作
  - 平台
    - Win32+python+pnuplot
    - Linux+python+pnuplot
  - 数据
    - Training Set
    - Test Set
  - SVM基础知识

样本文件格式

- 文本编码 ASCII/ANSI
- 数据存储格式
  - 每行格式：label feature1:value1 index2:value2 ...
    - label为类别标号，feature为特征序号，value为特征的值
    - value为0时该项可以省略（大规模数据时节省存储空间）
  - 示例：iris.tr（UCI / Iris Plant, 4 features, 3 classes）

```
1 1:-0.555556 2:0.5 3:-0.694915 4:-0.75
3 1:-0.166667 2:-0.333333 3:0.38983 4:0.916667
2 1:-0.333333 2:-0.75 3:0.0169491 4:-4.03573e-08
1 1:-0.833333 3:-0.864407 4:-0.916667
1 1:-0.611111 2:0.0833333 3:-0.864407 4:-0.916667
3 1:0.611111 2:0.333333 3:0.728813 4:1
3 1:0.222222 3:0.38983 4:0.583333
2 1:0.222222 2:-0.333333 3:0.220339 4:0.166667
2 1:-0.222222 2:-0.333333 3:0.186441 4:-4.03573e-08
…
```

- 格式检验脚本 checkdata.py

```
python checkdata.py iris.tr.txt
```

数据标准化

- 为何需要数据标准化?
  - 去除量纲
  - 简化运算

- 常见的标准化方法

极值标准化（libSVM）	归一化标准化	标准差标准化
$x^*_{i,j}=\frac{x_{i,j}-\mathrm{m}_j}{M_j-m_j}$	$x^*_{i,j}=\frac{x_{i,j}}{\sum_i x_{i,j}}$	$x^*_{i,j}=\frac{x_{i,j}-\bar{x}_j}{S_j}$

- 一些经验之谈
  - 训练集与测试集一起标准化!
  - 对于新来的测试集，怎么办?
  - 对于回归问题，量纲大的标签也需要标准化，此时预测值需要反标准化

LibSVM数据标准化

- svm-scale命令（使用的是极值标准化）
  - 格式：svm-scale [options] filename

    options:
    - 上下界（默认[-1,1]）：-l lower -u upper
    - 存储标准化尺度：-s scalefile
    - 加载标准化尺度：-r scalefile
    - 标签的标准化（用于回归）：-y lower upper
  - 使用提醒
    - 训练集和测试集一起scale，将所有数据缩放到[lower, upper]
    - 新来的测试样本，应该使用训练时候的标准化尺度进行标准化
  - 示例

```
svm-scale -s iris.scale iris.train
```

```
svm-scale -l -0.8 -u 0.8 -s iris.scale iris.train > iris.train.scaled
```

```
svm-scale -r iris.scale iris.test
```

LibSVM训练 - 1

- svm-train 命令
  - 格式: svm-train [options] filename [modelfile]

    options（部分）
    - -s svm_type : set type of SVM

0 (default)	1	2	3	4
C-SVC ☆☆	one-class SVM	One-class SVM	Epsilon-SVR ☆	Nu-SVR

    - -t kernel_type : set type of kernel function

0	1	2(default)	3
linear ☆	polynomial	radial basis function (RBF) ☆☆	sigmoid

    - -v n: n-fold cross validation mode
    - -g gamma (γ) : set gamma in kernel function (default 1/k)
    - -c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
    - -b probabilityestimates: whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
    - -m cachesize : set cache memory size in MB (default 100)

LibSVM训练 - 2

- 示例
  - 最简单的训练，所有参数均默认

```
svm-train iris.train
```

  - 任务类型默认(C-SVC)、核函数默认(RBF)，10-fold, c=100, g=0.01, 保存训练模型

```
svm-train -v 10 iris.train iris.model
```

  - 任务类型默认(C-SVC)，选用线性核函数，10-fold, c=100

```
svm-train -t 0 –v 10 –c 100 –g 0.01 iris.train iris.model
```

  - 采用概率模型，其余同上

```
svm-train –b 1 -t 0 –v 10 –c 100 –g 0.01 iris.train iris.model
```

- 一个重要问题，怎么选择参数？
  - 自动：libSVM提供的寻优脚本
  - 手动：**用试的！！**

LibSVM训练参数寻优

- 参数自动寻优的训练脚本 grid.py (Cross-validation and Grid-search)
  - 适用任务：分类问题 & RBF核(或linear核)
  - 格式：python [options] grid.py trainingfilename
  - options
    - -svmtrain pathname
    - -gnuplot pathname
    - -out pathname
    - -png pathname
    - -log2c begin,end,step
    - -log2g begin,end,step
    - additional same options for svm-train
  - 示例

```
python grid.py iris.train
```

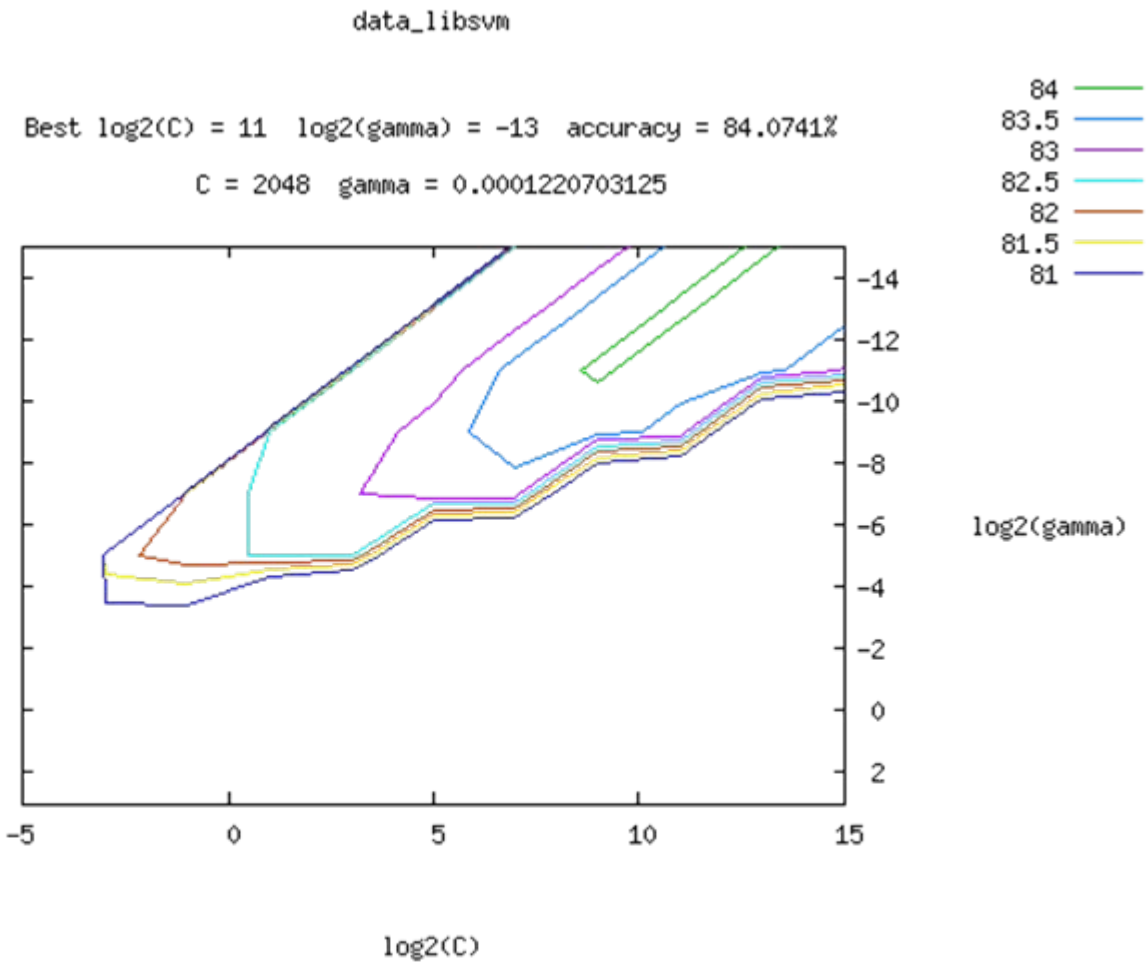
```
python grid.py –svmtrain d:\libsvm\svm-train.exe –gnuplot d:\gnuplot\bin\pgnuplot.exe
 -png d:\iris.gird.png –log2c -8,8,2 –log2g 8,-8,-2 –v 10 iris.train
```

  - linear核怎么使用？——设置一个虚拟的gamma参数: -log2g 1,1,1

```
python grid.py –log2c -8,8,2 –log2g 1,1,1 –t 0 –v 10 iris.train
```

寻优结果

- 参数寻优图示


data_libsvm
Best log2(C) = 11 log2(gamma) = -13 accuracy = 84.0741%
C = 2048 gamma = 0.0001220703125
84
83.5
83
82.5
82
81.5
81
-14
-12
-10
-8
-6
-4
-2
0
2
log2(gamma)
-5
0
5
10
15
log2(C)

LibSVM测试

- 格式： svm-predict [options] testfile modelfile resultfile
  options
  - -b probability : -b 0只输出类别； -b 1输出各类概率
- 返回结果
  - 各测试样本的类别（-b 1时为各类后验概率）
  - 正确率
- 示例

```
svm-predict iris.test iris.model iris.result
```

```
svm-predict –b 1 iris.test iris.model iris.result
```

- 值得注意的
  - 部分任务不支持概率输出（比如SVR, one-class SVM）
  - -b参数需要svm-train的时候建立概率模型与之对应

```
svm-train –b 1 iris.train iris.model
```

  - 新来的样本按照训练集标准化的尺度进行标准化

```
svm-scale -r iris.train.scale iris.test > iris.test.scaled
```

怎么用LibSVM做回归？

- 数据标准化
  - 对label同时进行标准化（量纲较小的时候可以忽略该步）

```
svm-scale -y -1 1 regression.train.scaled regression.model
```

- 参数寻优
  - 脚本：grid.py → gridregression.py
  - 寻优的参数：-c -g → -c -g –p

```
python gridregression.py –log2c -8,8,2 –log2g 8,-8,-2 –log2p -8,8,2 –v 10 regression.train
```

- 训练建模
  - 任务的选择：-s 3（epsilon - SVR）
  - 核函数的选择：通常选择 -t 2（RBF核，默认）或-t 0（linear核）
  - 与分类任务相比，多了一个基本参数p，建模时就用寻优找到的参数

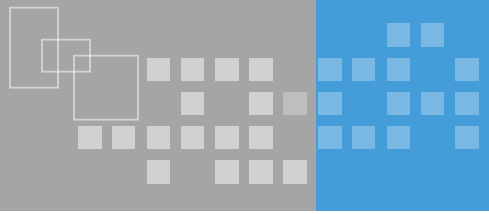
```
svm-train –s 3 -c 100 –g 0.01 –p 0.1 regression.train.scaled regression.model
```

- 测试
  - 评估指标：Rate → MSE

```
svm-predict regression.test regression.model regression.result
```

  - 返回值需要经过反标准化，恢复原来的量纲：好像要你自己做了！

Python平台下的libsvm - 1



- 安装
  - 将已经编译好的svmc.pyd文件（位于libsvm\windows\python\目录下）拷贝到系统环境变量目录（如python根目录）
  - 将svm.py文件拷贝到当前文件夹
  - 将cross_validation.py拷贝到当前文件夹
- 使用
  - 导入模块

```
from svm import *
```

  - 设置训练参数（包含svm_type, kernel_type, gamma, C…）

```
param = svm.svm_parameter(svm_type = svm.C_SVC, kernel_type = svm.LINEAR)
param.kernel_type = svm.RBF
```

  - 加载数据

```
ListLable = [1, -1]
ListValue = [[1, 0, 1], [-1, 0 ,-1]]
# ListValue = [{1:1, 3:1}, {1:-1, 3:-1}]
prob = svm_problem(ListLabel, ListValue)
```

Python平台下的libsvm - 2

- 建模

```
mod = svm_model(prob, param)
```

```
target = cross_validation (prob, param, n)
```

- 模型保存与加载

```
mod.save('modelfile')
```

```
mod2 = svm_model('modelfile')
```

- 测试

```
r = mod.predict ([1, 1, 1])
d = mod.predict_values([1, 1, 1])
prd, prb = m.predict_probability([1, 1, 1])
```

Thank you!

Q? / A