ctsm 关注此人
关注(0) 粉丝(6) 积分(9)
这个人很懒，啥也没写
.发送私信 .请教问题

博客分类

- 工作日志(6)
- 日常记录(0)
- 转贴的文章(0)

阅读排行

1. 1. 遗传算法
2. 2. KNN的一些总结
3. 3. K-means算法
4. 4. 字符串相关问题
5. 5. PageRank算法
6. 6. adaboost

最新评论

- @wmx3ng：非常感谢.你说的求聚类中心点算法,是通过求各维上... 查看»
- @Dennise：谢谢 查看»
- @Timco：不错，最近打算看看算法问题 查看»

访客统计

- 今日访问：35
- 昨日访问：53
- 本周访问：144
- 本月访问：1358
- 所有访问：15663

原 KNN的一些总结

发表于2年前(2014-01-19 12:00) 阅读（7021）| 评论（0）7人收藏此文章, 我要收藏

摘要 本文从自己的一些思考出发，总结了KNN的几个重要的问题，比较适合入门学习KNN

KNN 最近邻 k近邻 KD树 入门 误差 通俗 机器学习 实例

什么是KNN算法呢？顾名思义，就是K-Nearest neighbors Algorithms的简称。我们可能都知道最近邻算法，它就是KNN算法在k=1时的特例，也就是寻找最近的邻居。我们从名字可以知道我们要寻找邻居，但是为什么要寻找邻居，如何选取邻居，选取多少邻居，怎么样去寻找我们想要的邻居，以及如何利用邻居来解决分类问题这是KNN算法需要解决的几大问题，好了闲话不多说，进入正题。

首先我要说的是为什么我们要寻找邻居啊，古话说的好，人以类聚，物以群分，要想知道一个人怎么样，去看看他的朋友就知道了，其实这个过程就蕴含了KNN的算法核心思想，我们如果要判断一个样本点的类别，去看看和它相似的样本点的类别就行了， If it walks like a duck, quacks like a duck, then it is probably a duck，如图1所示：

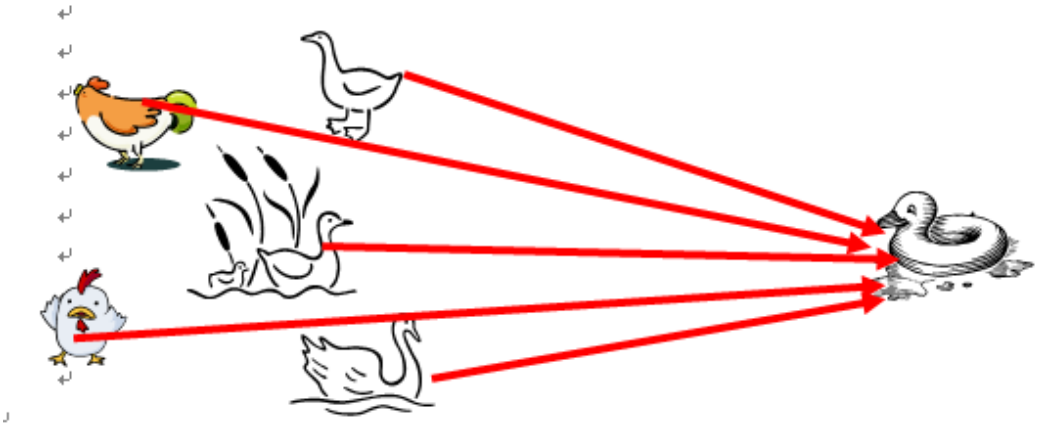


图 1

好了，在深入了解KNN之前有必要了解一下分类算法的大致情况以及其完整定义。图2所示的是一般的分类模型建立的步骤，分类一般分为两种：

积极学习法（决策树归纳）：先根据训练集构造出分类模型，根据分类模型对测试集分类。

消极学习法（基于实例的学习法）:推迟建模， 当给定训练元组时，简单地存储训练数据（或稍加处理），一直等到给定一个测试元组。

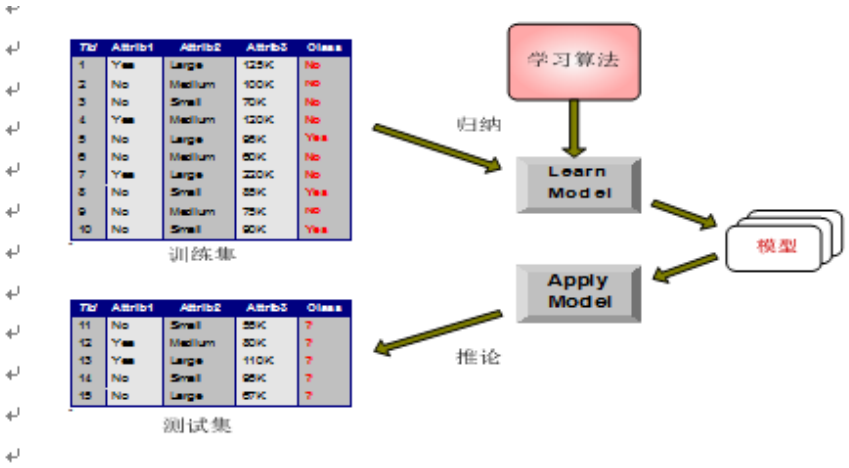


图 2

消极学习法在提供训练元组时只做少量工作，而在分类或预测时做更多的工作。KNN就是一种简单的消极学习分类方法，它开始并不建立模型，而只是对于给定的训练实例点和输入实例点，基于给定的邻居度量方式以及结合经验选取合适的k值，计算并且查找出给定输入实例点的 k 个最近邻训练实例点，然后基于某种给定的策略，利用这 k 个训练实例点的类来预测输入实例点的类别。算法的过程如图3所示：

Input : D, the set of training objects, the test object, $\mathbf{z}$, which is a vector of attribute values, and L, the set of classes used to label the objects

Output : $C_z \in L$, the class of z

For each object $\mathbf{y} \in D$ **do**

Compute $d(\mathbf{z}, \mathbf{y})$, the distance between $\mathbf{z}$ and $\mathbf{y}$;

end

Select $N \subseteq D$, the set (neighborhood) of k closest training objects for z;

$$C_z = \underset{v \in L}{argmax} \sum_{y \in N} I(v = class(cy));$$

where $I(\cdot)$ is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

图 3

了解了KNN的主体思想以后，接下来我们就来逐一的探讨和回答我在第一章所提出的四个问题，第一个就是如何度量邻居之间的相识度，也就是如何选取邻居的问题，我们知道相似性的度量方式在很大程度上决定了选取邻居的准确性，也决定了分类的效果，因为判定一个样本点的类别是要利用到它的邻居的，如果邻居都没选好，准确性就无从谈起。因此我们需要用一个量来定量的描述邻居之间的距离，也可以形象的表述为邻居之间的相似度，具体的距离度量方式有很多，不同的场合使用哪种需要根据不同问题具体探讨，具体的我就不罗嗦，在这篇博文http://www.cnblogs.com/v-July-v/archive/2012/11/20/3125419.html中有详细的阐述。以下给出了使用三种距离（欧式距离，曼哈顿距离，还有切比雪夫距离）的对glass数据集测试的例子，测试结果如图4所示：红线指的是实验使用的距离度量方式，黄线指的是实验的结果，可以看出使用曼哈顿距离分类效果明显好于其他两种。

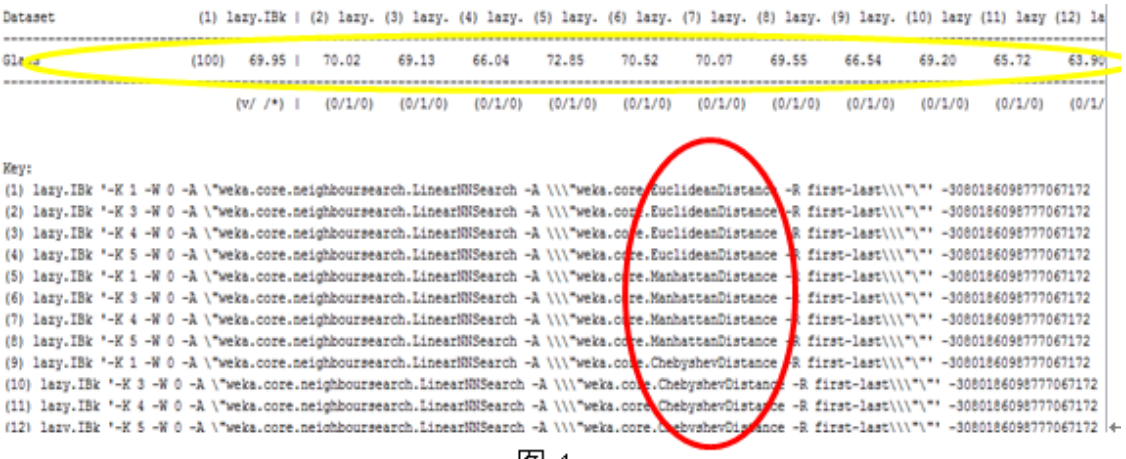


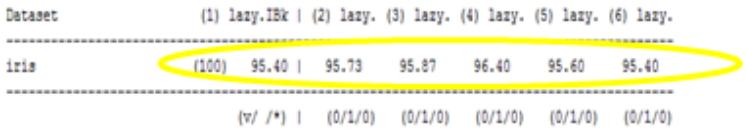
图 4

在给定了度量方式以后，我们自然而然会遇到一个问题就是到底要找多少个邻居才合适了，如图5所示 ，X是待分类样本， ‘+’ 和 ‘-’ 是样本类别属性，如果K选大了的话，可能求出来的k最近邻集合可能包含了太多隶属于其它类别的样本点，最极端的就是k取训练集的大小，此时无论输入实例是什么，都只是简单的预测它属于在训练实例中最多的累，模型过于简单，忽略了训练实例中大量有用信息。如果K选小了的话，结果对噪音样本点很敏感。那么到底如何选取K值，其实我在前面也说了，其实完全靠经验或者交叉验证（一部分样本做训练

(a) Neighborhood too small.　(b) Neighborhood just right.　(c) Neighborhood too large.

图 5

集，一部分做测试集）的方法，就是是K值初始取一个比较小的数值，之后不段来调整K值的大小来时的分类最优，得到的K值就是我们要的，但是这个K值也只是对这个样本集是最优的。一般采用k为奇数，跟投票表决一样，避免因两种票数相等而难以决策。下面我们可以通过交叉验证的方式求出最合适的K值，对iris数据（UCI Machine Learning Repository下载）用kNN算法进行分类，通过交叉验证（10次）的方式，对k取不同值时进行了实验，实验结果如图5所示，其中红线指的是实验选用的K值，黄线指的是实验的结果，我们发现在我所选取的k值中，当k=17时效果最好，在k=1时，即用最近邻来进行分类的效果也不错，实验结果呈现一个抛物线，与我们之前分析的结果相吻合。



Dataset	(1) lazy.IBk	(2) lazy.	(3) lazy.	(4) lazy.	(5) lazy.	(6) lazy.
iris	(100) 95.40	95.73	95.87	96.40	95.60	95.40
	(v/ /*)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)

Key:
(1) lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\" -3080186098777067172
(2) lazy.IBk '-K 5 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\" -3080186098777067172
(3) lazy.IBk '-K 11 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\" -3080186098777067172
(4) lazy.IBk '-K 17 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\" -3080186098777067172
(5) lazy.IBk '-K 23 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\" -3080186098777067172
(6) lazy.IBk '-K 29 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\" -3080186098777067172

图 6

好了，到这一步工作已经做了一半了，接下来就是如何去寻找这k个邻居了，因为对每一个待测样本点来说，我们都要对整个样本集逐一的计算其与待测点的距离，计算并存储好以后，接下来就是查找K近邻，这是最简单，也是最笨的方法，计算量太大了。因此KNN的一大缺点需要存储全部训练样本，以及繁重的距离计算量，有没有简单的一点的方法可以避免这种重复的运算啊，改进的方案有两个，一个是对样本集进行组织与整理，分群分层，尽可能将计算压缩到在接近测试样本邻域的小范围内，避免盲目地与训练样本集中每个样本进行距离计算。另一个就是在原有样本集中挑选出对分类计算有效的样说本，使样本总数合理地减少，以同时达到既减少计算量，又减少存储量的双重效果。KD树方法采用的就是第一个思路，关于KD树及其扩展可以参看博文http://www.cnblogs.com/v-July-v/archive/2012/11/20/3125419.html，它对其进行了详细的阐述，我就不啰嗦了。我想补充的是压缩近邻算法，它采用的思路是第二种方案，利用现有样本集，逐渐生成一个新的样本集，使该样本集在保留最少量样本的条件下，仍能对原有样本的全部用最近邻法正确分类，那么该样本集也就能对待识别样本进行分类，并保持正常识别率。它的步骤如下：

首先定义两个存储器，一个用来存放即将生成的样本集，称为Store；另一存储器则存放原样本集，称为Grabbag。其算法是：

1. 初始化。Store是空集，原样本集存入Grabbag；从Grabbag中任意选择一样本放入Store中作为新样本集的第一个样本。

2.　　样本集生成。在Grabbag中取出第i个样本用Store中的当前样本集按最近邻法分类。若分类错误，则将该样本从Grabbag转入Store中，若分类正确，则将该样本放回Grabbag中。

3.　　结束过程。若Grabbag中所有样本在执行第二步时没有发生转入Store的现象，或Grabbag已成空集，则算法终止，否则转入第二步。

当然解决的方案很多，还有比如剪辑近邻法，快速搜索近邻法等等很多，就不一一介绍了。下面测试了一下不同最近邻搜索算法（线性扫描，kd树，Ball树，Cover树）所花费的时间，如表1所示：

算法	LNN	kd	BALL	Cover
时间（s）	686	174	241	168

表 1

到这一步基本上是万事俱备，只欠东风啦。K近邻（通俗的来说就是某人的k个最要好的朋友都找出来啦）都求出来啦，接下来就是要朋友们利用手中的投票器为其投票啦。一般的做法就是一人一票制，少数服从多数的选举原则，但是当和我测试对象离的近的数量少，而离得远的数量多时，这种方法可能就要出错啦，那咋办呢，看过歌唱选秀节目的人应该清楚，评审分为两种，一种是大众评审一人一票，一种是专家评审，一人可能有很多票，我们也可以借鉴这个思想，为每个邻居赋予一定的投票权重，通过它们与测试对象距离的远近来相应的分配投票的权重,最简单的就是取两者距离之间的倒数$1/d(y,z)^2$，距离越小，越相似，权重越大，将权重累加后选择累加值最高类别属性作为该待测样本点的类别。我用不同的权重方式对UCI中的glass数据集进行测试，图7显示的是直接不采用权重的实验结果，图8显示的是权重为距离的倒数，图9显示的是权重为1减去归一化后的距离，红线指的是实验使用的权重赋值方式，“0”指的是不采用权重，“0 -I”指的是取距离倒数，“0-F”指的是1减去归一化后的距离，深红线指的是实验的结果，我们可以看出采用了权重的总体上来说比不使用权重要好。

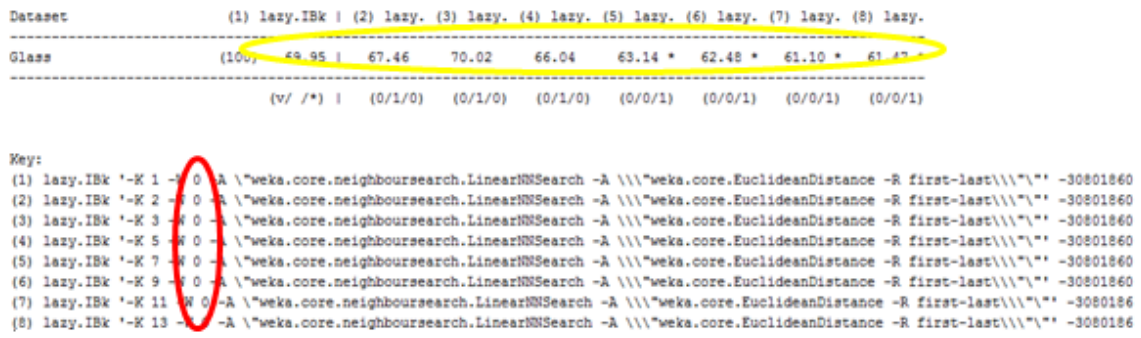


图 7

```
Dataset                  (1) lazy.IBk | (2) lazy. (3) lazy. (4) lazy. (5) lazy. (6) lazy. (7) lazy. (8) lazy. (9) lazy.
-------------------------------------------------------------------------------------------------------------------------
Glass                    (100)  69.95 |  69.95    71.32    70.32    71.26    69.99    68.05    67.76    66.65
-------------------------------------------------------------------------------------------------------------------------
                              (v/ /*) |  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)

Key:
(1) lazy.IBk '-K 1 -W 0 -I -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -3080186098
(2) lazy.IBk '-K 2 -W 0 -I -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -3080186098
(3) lazy.IBk '-K 3 -W 0 -I -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -3080186098
(4) lazy.IBk '-K 4 -W 0 -I -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -3080186098
(5) lazy.IBk '-K 5 -W 0 -I -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -3080186098
(6) lazy.IBk '-K 7 -W 0 -I -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -3080186098
(7) lazy.IBk '-K 9 -W 0 -I -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -3080186098
(8) lazy.IBk '-K 11 -W 0 -I -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -308018609
(9) lazy.IBk '-K 13 -W 0 -I -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -308018609
```

图 8

```
Dataset                  (1) lazy.IBk | (2) lazy. (3) lazy. (4) lazy. (5) lazy. (6) lazy. (7) lazy. (8) lazy. (9) lazy.
-------------------------------------------------------------------------------------------------------------------------
Glass                    (100)  69.95 |  69.95    70.76    70.42    68.74    67.16    64.31    64.28    62.96 *
-------------------------------------------------------------------------------------------------------------------------
                              (v/ /*) |  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)  (0/1/0)  (0/0/1)

Key:
(1) lazy.IBk '-K 1 -W 0 -F -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -308018609
(2) lazy.IBk '-K 2 -W 0 -F -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -308018609
(3) lazy.IBk '-K 3 -W 0 -F -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -308018609
(4) lazy.IBk '-K 4 -W 0 -F -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -308018609
(5) lazy.IBk '-K 5 -W 0 -F -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -308018609
(6) lazy.IBk '-K 7 -W 0 -F -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -308018609
(7) lazy.IBk '-K 9 -W 0 -F -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -308018609
(8) lazy.IBk '-K 11 -W 0 -F -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -30801860
(9) lazy.IBk '-K 13 -W 0 -F -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"" -30801860
```

图 9

至此关于KNN算法的描述就到此结束了。可以看出算法的思想是十分简单的，我们自然而然的就会想这个算法的准确率到底是多少，有没有啥科学的证明，其实最初的近邻法是由Cover和Hart于1968年提出的，随后得到理论上深入的分析与研究，是非参数法中最重要的方法之一，它在论文Nearest Neighbor Pattern Classification中给出了算法准确率的详细描述。最近邻法的错误率是高于贝叶斯错误率的，$P^* \le P \le P^*(2-\frac{C}{C-1}P^*)$ 其中 P^* 代表的是贝叶斯误差率，由于一般情况下P*很小，因此又可粗略表示成：$P^* \le P \le 2P^*$，对于kNN来说，当样本数量N→∞的条件下，k-近邻法的错误率要低于最近邻法，具体如图10所示：

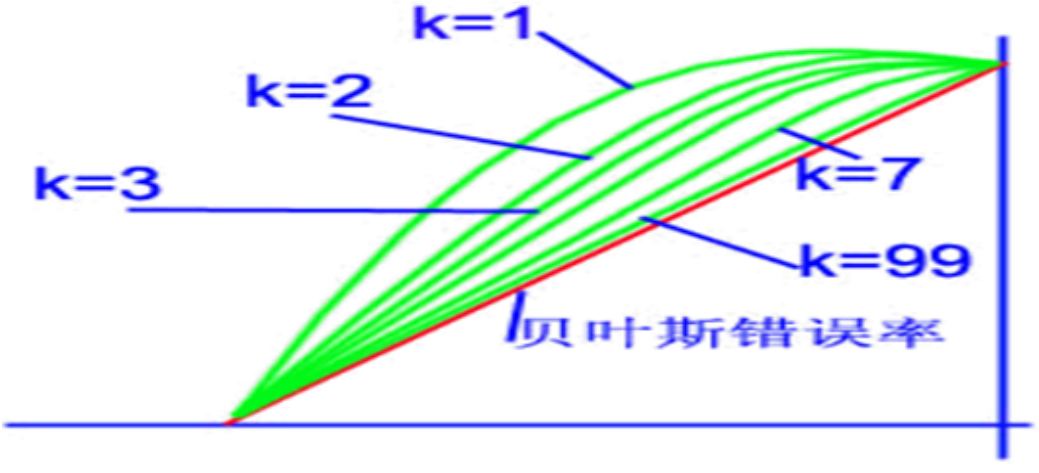


图 10

最后来一下小小的总结，KNN是典型的非参数法，是一个非常简单，性能优秀的分类算法,人们正在从不同角度提出改进KNN算法,推动了KNN算法的研究工作,使KNN算法的研究得到了快速的发展。

本文行文仓促，加之水平有限，希望得到大家的批评指正，多谢！！！

2014年1月19日中午 by ido

分享到： 新浪微博 weibo.com 腾讯微博 t.qq.com _3赞

- « 上一篇
- 下一篇 »

评论0

插入： 表情 开源软件

发表评论

关闭插入表情

关闭相关文章阅读

- 2015/02/28 机器学习之分类算法一：K-近邻算法
- 2015/01/03 关于机器学习的学习笔记（三）：k近...
- 2015/12/10 机器学习之K近邻算法（KNN）
- 2014/06/13 kNN：k-nearest neighbor classifi...
- 2015/09/24 k近邻算法