摘 要

随着 Internet 的迅速发展，网络攻击种类不断演化更新，发生频率持续增长。由于传统的基于模式匹配的入侵检测系统不能发现未知攻击并且需要不断升级的完备攻击特征库，其发展有很大的局限性。因此目前网络业务量数据的入侵检测技术的研究重点转向了异常检测。

本文将一种广泛用于多维空间最邻近搜索的数据结构，kd 树，应用于网络数据的异常检测。首先对 kd 树建立算法、分割策略进行了描述。然后，设计并实现了一个简单的基于 kd 树的异常检测系统。最后通过实验给出了系统对不同种类攻击的检测效果。

我们在 KDD1999 数据集的基础上构造了训练数据集和测试数据集，依照 Wenke Lee 的检测属性集理论对系统进行了训练和测试。通过实验，我们找到了系统最佳的偏差度量方式。同时，实验结果证时了将 kd 树应用于异常检测领域的可行性及 Wenke Lee 理论的正确性。

关键词 入侵检测；异常检测；kd 树；检测属性集

Abstract

With the rapid development of Internet, the types of attacks against computer networks evolved continuously. The traditional intrusion detection systems based on pattern matching are limited by its incapability to detect new attack types and their need of an up-to-date attack feature database. The current research on network intrusion detection has mostly moved to anomaly detection.

In this paper,we apply kd-tree,a popular data structure for nearest neighbor searching in multidimensional space,into network anomaly detection. Firstly We demenstrate the building algorithm and the splitting method.Then a simple intrusion detection system based on kd-tree is build. At last,we give the detection results of the system for different attacks by some experiments.

We construct the training dataset and the test dataset based on KDD1999.Then ,system is tested in terms of the detection attributes theory of Wenke Lee.According to experiments,we find the best methord to measure the difference between the data point and the profile.Additonally，the experiment resuls demonstrate the feasibility of intrudrucing kd-tree into anomaly detection and the correctness of Wenke Lee's theory.

Key words Intrusion Detection; Anomaly Detection; kd-tree; detection attributes theory

目 录

第1章 绪 论

1.1 课题背景

随着计算机技术的迅速发展，在计算机上处理的业务也由基于单机的数学运算、文件处理，基于简单连接的内部网络的业务处理、办公自动化等发展到基于企业复杂的内部网（Intranet）、企业外部网（Extranet）、全球互连网（Internet）的企业级计算机处理系统和世界范围内的信息共享和业务处理。在系统处理能力提高的同时，系统的连接能力也在不断的提高。但在连接能力、流通能力提高的同时，基于网络连接的系统安全问题也日益突出。

据Warroon Research的调查，1997 年世界排名前一千的公司几乎都曾被黑客闯入。据美国FBI 统计，美国每年因网络安全造成的损失高达75 亿美元。计算机紧急响应小组（CERT）的年度报告列出了1995 年发生的将近2500 个有报道的安全事故，其影响涉及到1200 个Internet 站点。我国的ISP、证券公司及银行也多次被国内外黑客攻击。如上海市复旦大学一学生曾通过电话拨号系统非法侵入上海几个著名的网络站点，破解了大部分的系统帐号。这还仅仅是来自系统外部的通过Internet 进行的系统入侵。实际上在系统资源损失中，更大一部分的威胁来自系统内部，据Ernst 和Young 统计，来自系统内部的攻击行为在整个系统受到的攻击中占到了70%以上。

事实上，现代计算机系统从诞生至今一直都面临着各类安全隐患的侵扰。无论是UNIX 缓冲区溢出还是Microsoft Internet Explorer的Bug 问题，安全漏洞总是到处出现在应用程序和操作系统的各个层次上。

入侵的威胁多是通过挖掘操作系统和应用服务程序的弱点或者缺陷来实现。现实生活中，网络滥用案例已经如同计算机网络本身的发展一样在全世界范围内得到迅速蔓延。各类异构的计算机网络、操作系统、通信协议和网络应用的增殖也给入侵检测带来了很多困难与挑战。网络互连能力的增加也为外部入侵者提供了更多的访问机会，并使之更易于逃避身份识别。对于以上提到的问题，很多组织正在致力于提出更多的更强大的主动策略和方案来增强网络的安全性，然而另一个更为有效的解决途径就是入侵检测。在入侵检测之前，大量的安全机制都是从主观的角度设计的,他们没有根据网络攻

击的具体行为来决定安全对策，因此，它们对入侵行为的反应非常迟钝，很难发现未知的攻击行为，不能根据网络行为的变化来及时地调整系统的安全策略。而入侵检测正是根据网络攻击行为而进行设计的，它不仅能够发现已知入侵行为，而且有能力发现未知的入侵行为，并可以通过学习和分析入侵手段，及时地调整系统策略以加强系统的安全性。

1.2 入侵检测概述

为了保护Internet上信息、服务和访问的安全性，人们开发了多种安全协议和技术，主要有加密技术、用户认证技术、安全协议、防火墙技术和入侵检测技术等。加密技术、用户认证技术通常当成保护计算机系统的第一道防线，而入侵检测技术则是在防火墙之后的第二道安全屏障，它是构架网络安全系统的一个重要环节。入侵检测技术在不影响整个网络性能的前提下，对网络进行监控。本节将总览入侵检测技术，并介绍这项技术中的相关概念。

1.2.1 入侵检测的定义

所谓入侵检测就是指通过行为、安全日志、审计数据或其它网络上呆以获得的信息进行操作，检测到对系统的闯入或闯入的企图。这里所说的入侵是指任何破坏网络信息或资源的机密性、完整性、可用性和真实性的行为。入侵检测不仅检测来自外部的的入侵行为，同时也指出内部用户的未授权活动。它应用了以攻为守的策略，所提供的数据不仅有可能用来发现合法用户是否滥用特权，还有可能在一定程度上提供追究入侵者法责任的有效证据。

入侵检测系统可以被定义为对计算机和网络资源上的恶意使用行为进行识别和响应的处理系统。它通过对计算机系统进行监视，提供实时的入侵检测并采取相应的防护手段。入侵检测系统的目的在于检测可能存在的攻击行为。

1.2.2 入侵检测的起源

入侵检测的概念早由Anderson在1980年提出[ADS80]，他提出了入侵检测系统的三种分类方法。Denning对Anderson的工作进行了扩展，她详细探

讨了基于异常和误用检测方法的优缺点，于1987年提出了一种通用的入侵检测模型[DEN87]。这个模型独立于任何特殊的系统、应用环境、系统脆弱性和入侵种类，因此提供了一个通用的入侵检测专家系统框架，并有IDES原型系统的实现。

IDES原型系统采用的是一个混合结构，包含了一个异常检测器和一个专家系统。异常检测器采用统计技术刻划异常行为，专家系统采用基于规则的方法检测已知的危害安全的行为。异常检测器对行为的渐变是自适应的，因此引入专家系统能有效防止逐步改变的入侵行为，提高准确率。该模型为入侵检测技术的研究提供了良好的框架结构，为后来各种模型的发展奠定了基础，导致了随后几年内一系列系统原型的研究。如Discovery、Haystack、MIDS、NADIR、NSM、Wisdom and Sense等。

一个较为通用的入侵检测模型如图 1-1 所示：

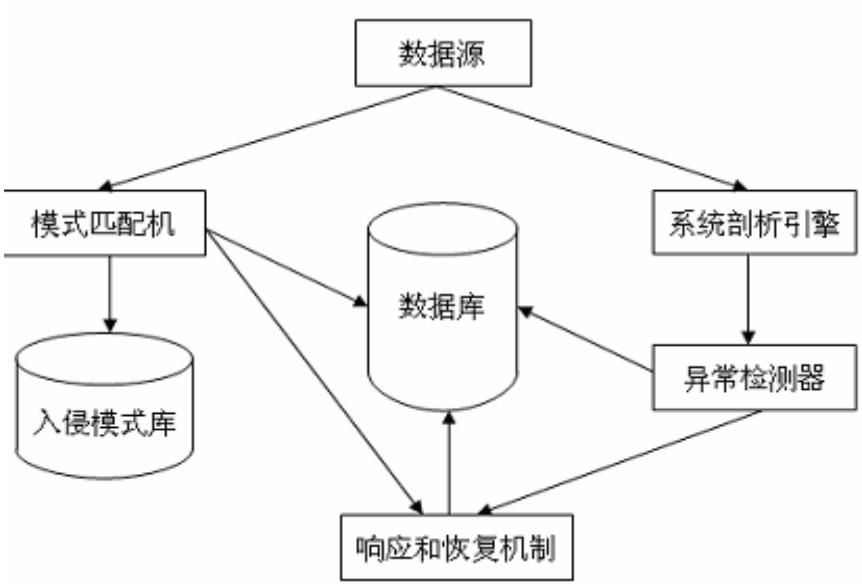

图 1-1

1.2.3 入侵检测的分类

入侵检测有三种分类方法，其一是根据其采用的技术分类，可以分为：

（1）异常检测（Anomaly Detection）

进行异常检测的前提是认为入侵是异常活动的子集。异常检测系统通

过运行在系统或应用层的监控程序监控用户的行为，通过将当前主体的活动情况和用户轮廓进行比较。用户轮廓通常定义为各种行为参数及其阀值的集合，用于描述正常行为范围。当用户活动与正常行为有重大偏离时即被认为是入侵。如果系统错误地将异常活动定义为入侵(false positive)，称为错报；如果系统未能检测出真正的入侵行为则称为漏报(false negative)。这是衡量入侵检测系统性能很重要的两个指标。

异常检测系统的效率取决于用户轮廓的完备性和监控的频率。因为不需要对每种入侵行为进行定义，因此能有效检测未知的入侵。同时系统能针对用户行为的改变进行自我调整和优化，但随着检测模型的逐步精确，异常检测会消耗更多的系统资源。

一个异常检测的模型如图 1-2 所示：

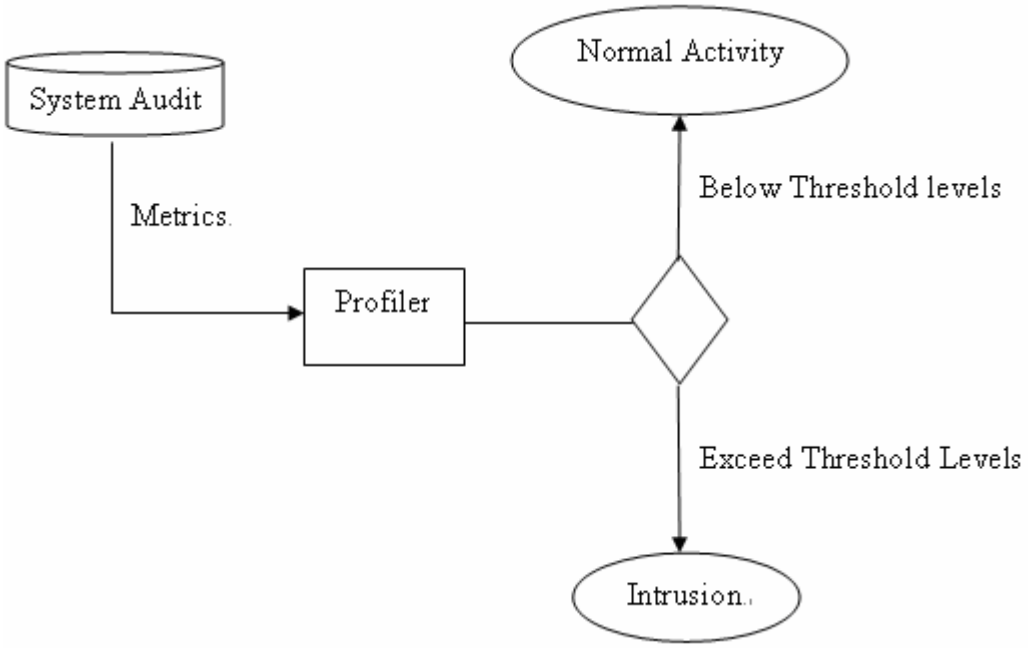

图 1-2

(2)误用检测（Misuse Detection）

进行误用检测的前提是所有的入侵行为都有可被检测到的特征。误用检测系统提供攻击特征库，当监测的用户或系统行为与库中的记录相匹配时，系统就认为这种行为是入侵。如果入侵特征与正常的用户行能匹配，则系统会发生错报；如果没有特征能与某种新的攻击行为匹配，则系统会发生漏报。

采用特征匹配，误用模式能明显降低错报率，但漏报率随之增加。攻击特征的细微变化，会使得误用检测无能为力。

一个误用检测的模型如图1-3所示：

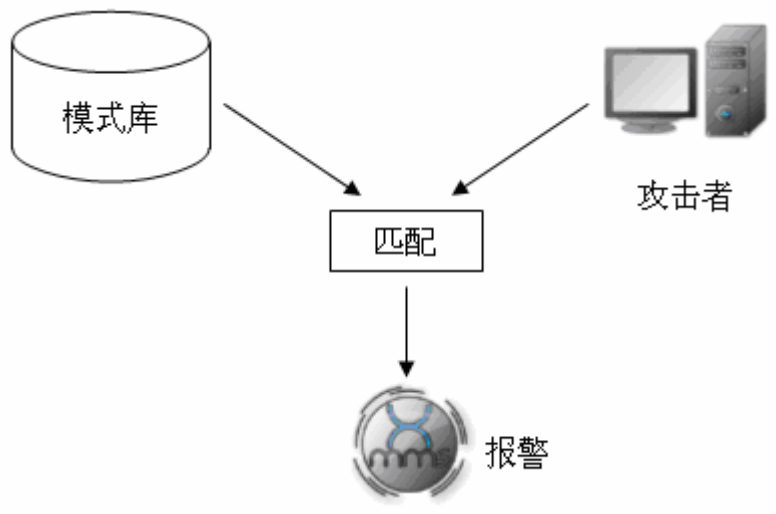

图 1-3

(3)特征检测（Specification-based Detection）

特征检测关注的是系统本身的行为。定义系统行为轮廓，并将系统行为与轮廓进行比较，对未指明为正常行为的事件定义为入侵。特征检测系统常采用某种特征语言定义系统的安全策略。这种检测方法的错报与行为特征定义准确度有关，当系统特征不能囊括所有的状态时就会产生漏报。特征检测最大的优点是可以通过提高行为特征定义准确度和覆盖范围，大幅度降低漏报和错报率；最大的不足是要求严格定义安全策略，这需要经验和技巧，另外为了维护动态系统的特征库通常是很耗时的事情。

另一类分类方法是根据系统所检测的对象是主机还是网络来分：

(1)基于主机的入侵检测系统

基于主机的入侵检测系统通过监视与分析主机的审计记录检测入侵。这些系统的实现不全在目标主机上，有一些采用独立的外围处理机，如Haystack。另外NIDES[AFV95]使用网络将主机信息传到中央处理单元。但它们全部是根据目标系统的审计记录工作。能否及时采集到审计记录是这些系统弱点之一，从而有入侵者会将主机审计子系统作为攻击目标以避开入侵检测系统。

(2)基于网络的入侵检测系统

基于网络的入侵检测系统通过在共享网段上对通信数据进行侦听采集

数据，分析可疑现象。与主机系统相比，这类系统对入侵者而言是透明的。由于这类系统不需要主机提供严格的审计，因而对主机资源消耗少，并且由于网络协议是标准的，它可以提供对网络通用的保护而无需顾及异构主机的不同架构。

按照控制方式,入侵检测系统的可以分类以下两类:

(1)集中式控制

一个中央节点控制系统中的所有入侵检测要素。集中式控制要求在系统组件间提供保护消息的机制，能灵活方便地的启动和终止组件，能集中控制状态信息并将这些信息以一种可读的方式传给最终用户。

(2)与网络管理工具相结合

将入侵检测简单地看作是网络管理的子功能。网管软件包搜集的一些系统信息流可以作为入侵检测的信息源。因此将这两个功能集成倒一起，便于用户使用。

1.3 入侵检测的研究现状

最初的网络入侵检测方法都是基于攻击特征的，通过对这些攻击特征进行识别和归纳来得出安全结论。但由于现有的网络系统存在安全性方面的先天缺陷，而且人们对攻击的认识总跟不上攻击的发展，所以基于这种思路来完全社绝网络入侵是不可能的。因此目前大规模网络入侵检测技术方面的研究重点转向了异常检测，即发展检测和抵御未知类型攻击的能力。

在基于知识发现的异常检测中的大部分研究都可以说是在尝试使用不同的建模方法。这些方法试图在正常数据之上构建一些模型，然后察看新的数据是否适应这个模型。例如：

Smart Shifter[22]是一个基于无监督学习方法的孤立点检测系统。这个系统是基于有限混合模型构建的。检查每一个新输入的数据相对于正常模式的偏离程度。与些同时，使用一个在线学习算法来保持对模型的更新。对于每一个新输入的数据都有一个数值来衡量学习之后的变化。如果数值过高，则意味着这个数据点为孤立点。

Parzen Window 网络入侵检测器[11]。该分类器采取的是基于Parzen Window 预测器非参量化密度预测方法，这种检测方法不需要训练，但同时也存在着检测时间无法接受的缺陷。

在[12]一文中，作者提出一种算法，人为的产生异常来迫使感应学习器

来找出已知类别（正常数据与已知攻击）之间的界限。他们在KDD99数据集上的实验结果表明，此种模型能够检测出77%的未知入侵类别。

除此之外，Lane 和 Brodley[25]通过察看用户行为模式以及比较入侵时的行为和正常使用时的行为，在未标记的数据上进行异常检测。在[8]中采用了使半增量技术，以产生用户行为模式的相似方法。[23]使用神经网络得到了模型，[24]使用非稳定数据模型来检测新的入侵。在 SRI 的 EMERALD 系统[8]中开发的技术，使用历史纪录作为它的正常训练数据。然后它比较了新数据分布与从历史数据中得到数据分布，数据分布之间的不同说明了攻击。

目前大多数 IDS 产品都只使用了入侵检测的部分方法，这导致攻击者能采用新的攻击方法轻易突破入侵检测系统的保护。入侵检测系统能在一些特定环境和配置下提供有效的保护手段，这其中涉及的关键方法是在关键子网和主机上感知和判断能力。下面是传统的入侵检测面临以下的问题：

(1)随着能力的提高，入侵者会研制更多的攻击工具，以及使用更为复杂精致的攻击手段，对更大范围的目标类型实施攻击；

(2)入侵者采用加密手段传输攻击信息；

(3)日益增长的网络流量导致检测分析难度加大；

(4)缺乏统一的入侵检测术语和概念框架；

(5)不适当的自动响应机制存在着巨大的安全风险；

(6)存在对入侵检测系统自身的攻击；

(7)过高的错报率和误报率，导致很难确定真正的入侵行为；

(8)采用交换方法限制了网络数据的可见性；

(9)高速网络环境导致很难对所有数据进行高效实时分析。

目前入侵检测领域要解决的一些难点包括：

(1)更有效的集成各种入侵检测数据源，包括从不同的系统和不同的传感器上采集的数据，以减少虚假报警；

(2)在事件诊断中结合人工分析；

(3)提高对恶意代码的检测能力，包括 email 攻击，Java，ActiveX 等；

(4)采用一定的方法和策略来增强异种系统的互操作性和数据一致性；

(5)研制可靠的测试和评估标准；

(6)提供科学的漏洞分类方法，尤其注重从攻击客体而不是攻击主体的

观点出发；

(7)提供对更高级的攻击行为如分布式攻击、拒绝服务攻击等的检测手段；

1.4 本文的研究内容及篇章节构

为了解决上述问题，我们提出了基于骨干路由器的分布式入侵检测系统（Global Router based Distributed Intrusion Detection System DIDS)。DIDS 具有以下特征：基于零拷贝的高效捕包平台；在线的基于统计的异常检测机制；基于归纳推理的异常检测机制；可升级的多数据源异常/入侵警报融合。同时由于入侵检测技术是一种被动的防御技术，在本文中加入了主动发现网络中可能存在漏洞和弱点技术的研究，增加了入侵检测系统的主动特性。

本文第一章概要介绍了入侵检测的定义、起源、分类，并简单总结了其研究现状。第二章设计了一个大流量下 SYN-Flooding 的异常检测算法。第三章研究了端口扫描技术并，并设计了一个可实现高效扫描的系统。第四章在前面研究的基础上设计并完成了一个分布式入侵检测系统。

1.5 本章小结

本章叙述了论文课题研究的目和意义，对入侵检测及其现状做了简要介绍，最后给出了论文的主要研究内容及章节安排。

第2章 kd 树

2.1 概述

2.1.1 kd 树的一般描述

Kd树又被称为多维二进制搜索树，由Bentley于1975提出，被广泛用于高维空间的数据索引和查询。从本质上说kd树是一种二元搜索树，它通过超平面把一个空间递归划分为多个子空间来实现二叉搜索。

Kd树的每个节点对应于多维空间的一个矩形，我们将其称为“盒子”。根节点对应的盒子包含了所有的数据点，中间节点则对应于被划分后的数据点的子集。超平面将节点不断进行划分，当节点中的数据个数小某个事先设定阈值（bucket size）时，划分就会停止，此时我们得到的节点被称为叶节点，所有数据最终都存放在叶节点中。

2.1.2 kd 树的形式化描述

对于一棵kd树,K维的欧氏空间被一个正交于任意一个K 维坐标轴的超平面分割为两个半空间。这个超平面可以表示为:

$$H = \{ | \ x \in RK \ ; \ xj = h \ | \ \}$$

被超平面分割的两个半空间分别为Rl 和Rr ，表示为:

$$Rl = \{ | \ x \in RK \ ; \ xj \leq h \ | \ \} \text{ 和 } Rr = \{ | \ x \in RK \ ; \ xj > h \ | \ \}$$

其中，j 为超平面的索引号；h 为超平面在坐标轴上的位置。

通过KD-tree ，可以把空间分布的矢量依据超平面进行二进制的划分，超平面的划分是一个连续递归进行的过程,而超平面划分的深度也就是kd树的深度，该划分深度用d 表示。

2.1.2 一个 kd 树的例子

下面的例子将使我们对kd树有一个感性的认识。

例：假设有二维空间中的8个点，我们设定每个叶节点存放两个数据，采用“标准分割策略”（见2.3.1）将其划分，得到kd树。

图2-1为划分前点集的分布：

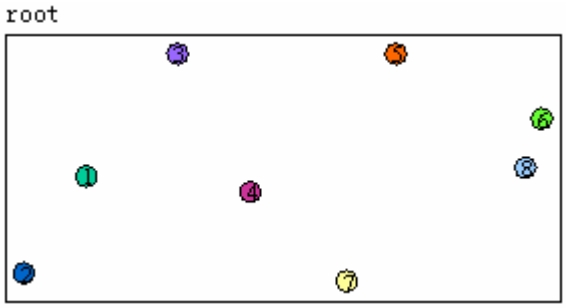

图2-1

图 2-2 为采用标准分割策略对点集进行划分后得到的结果，其中 Node1 为根节点，Node2、Node3 为分裂节点，Node4～Node7 为叶节点，H1～H3 为用于将点集进行划分的超平面：

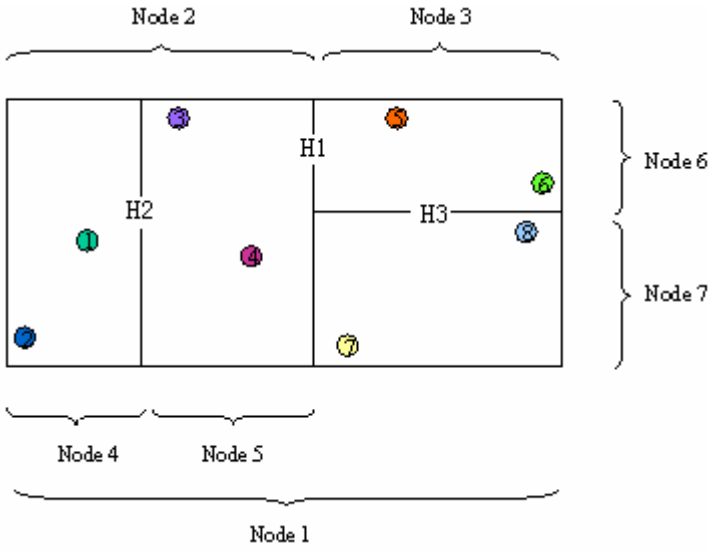

图2-2

图2-3为点集划分后得到的树形结构的直观图：

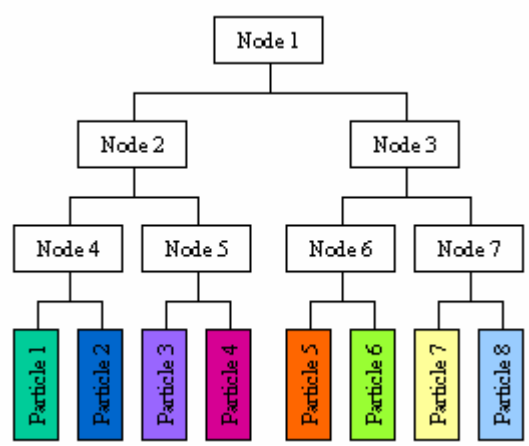

图2-3

2.2 kd 树的建立

Kd树的建立过程实际上就是按照一定的划分策略，用超平面将多维空间中的点集进行递归划分的过程。

为直观起见，我们以二维空间中的点为例，以2.1.3中的例子为背景，简要说明kd树的建立过程。

如图2-1所示，最初我们有二维空间中的8个点，它们在根节点Node1中。按照“ 标准分割策略 ”，我们用H1将其划分为Node2、Node3两部分（其中Node2含有数据点1~4，Node3含有数据点5～8）。因为我们设定每个节点的bucet size为2，而Node2、Node3中含有的节点数为4，所以需要继续对其进行划分。依然按照“ 标准分割策略 ”，我们选择用H2对Node2进行划分，得到Node4、Node5，此时每个节点中只含2个数据点，不大于bucket size值，因为无需对其进行进一步划分。用同样的方法对Node3进行处理，我们得到Node6,Node7。至此，我们建好了一棵kd树。

以下是对kd树建立过程的伪代码描述[7]，由7~10行，我们行容易看出，树的建立采用的是一种递归的算法：

Algorithm BUILDKDTREE(*P*,*depth*)

Input: A set of points P and the current depth *depth*.

Output: The root of a kd-tree storing P.

1. **if** P contains only one point **then**
2. **return** a leaf storing this point
3. **else if** depth is even **then**
4. Split P into two subsets with a vertical line l through the median x-coordinate of the points in P. Let P1 be the set of points to the left and P2 be the set of points to the right. Points exactly on the line belong to P1.
5. **else**
6. Split P into two subsets with a horizontal line l through the median y-coordinate of the points in P. Let P1 be the set of points above l and P2 be the points below l. Points exactly on the line belong to P1.
7. V_{right} := BUILDKDTREE(P1,depth+1)
8. V_{left} := BUILDKDTREE(P2,depth+1)
9. Create a node V with V_{right} and V_{left} as its right and left children, respectively.
10. **return** V.

这段代码它采用的分割策略是Bentley在最初提出kd树时使用的数据划分法（代码的3~6行是其分割策略的体现）。由于这种方法只是按照节点的层数对其包含的数据点进行机械的划分，而并没有考虑数据点在多维空间的分布，因此按照这种方法建起的kd树查询效率较低，且不适用于对数据进行聚类。但作为kd树原创者建树思想的一种体现，我们依然把它展示在这里。

2.3 分割策略

在kd树的建立过程中，分割策略起着至关重要的作用，它直接决定了kd树的形态，对同一数据集，如果采用不同的分割策略对其进行划分，得到的kd树可能有很大差别。同时，分割策略的选用，也将影响到建树后对树中数据点的询效率。

在对分割策略进行具体介绍之前，我们作如下定义：

（1）符号定义：

S:存储在树中的当前数据点的子集。

C:当前的盒子。

n:S中数据点的个数。

R(C)：当前盒子所在的超矩形。S中的点全部包含在R(C)中。

起初，S为全体数据点的集合，C是kd树的根节点，R(C)是包含了S中所有点的超矩形。

（2）概念：

长宽比：超矩形最长边与最短边的比值。

平均长宽比：树中所有超矩形的长宽比的平均值。

点集延展度：指定多维空间上的某一维，点集中的点在该维的上最大坐标与最小坐标的差称为点集在上的点集延展度。

分割维：在某一次分割中被选中的进行点集分割的维。

例如：有多维点（3,6,1）、（10,8,9）、（12，16,19）。考察其第一维的三个坐标，分别为3、10、12。因此，点集在第一维上的延展度为12（最大值）－3（最小值）＝9。同理，点集在第二、第三维上的延展度分别为10和18。

分割时有所要解决的问题主要有两个：一个是沿哪一维空间对点集进行分割（选择分割维），二是在该维的哪一点处放置超平面（选择分割点）。对这两个问题的解决方法不同，所得到的分割策略便不同，下面我们将介绍几种常见的分割策略。

2.3.1 标准分割策略

将 S 中具有最大延展度的维定为分割维，将该维的中点定为分割点。这种策略可以保证最终得到的 kd 树的高度不超过 $\log_2 n$。缺点是，可能使划分后得到的超矩形具有过高的长宽比。

例：设有二维空间的点集，存在于 R(C)中，如图 2-4 所示。

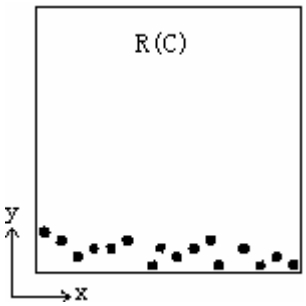

图 2-4

设定 bucket size 为 1，按照标准分割策略对其进行划分。显然点集最初在 x 方向有最大的延展度，设 x 为第 1 维，则分割维为 1，分割点坐标为 R(C)在 x 方向的中点，这样我们就得到了第一次划分的超平面 H。对用 H 划分后得到的两个子集进行同样的递归划分，最终我们得到如图 2-5 所示的 kd 树。

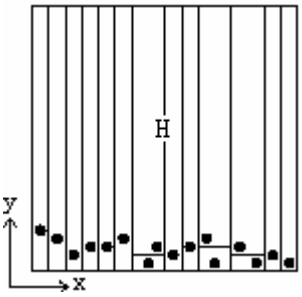

图2-5

当数据点在某些维上很紧凑，在另一些维上很稀疏时，就会使最终存放数据点的叶节点对应的超矩形有过高的长宽比。图 2-4 中所示的数据分布就是这样一种情况，相应的，我们最终得到的 kd 树中的超矩形的长宽比都很高，这一点从图 2-5 上容易看出。

2.3.2 中点分割策略

将超矩形的最长边所在的维定为分割维，如果最长边不止一条，则看点集沿哪条边有较大的延展度，将有最大延展度的边所在的维定为分割维（若延展度也相同，则从中任选一条边所在的维定为分割维即可），将此边的中点定为分割点。

按这种策略划分后得到的kd树，长宽比不会超过2。缺点是，可能产生较多的空叶节点。

对图2-4中的点集采用中点分割策略后得到的kd树如图2-6所示。

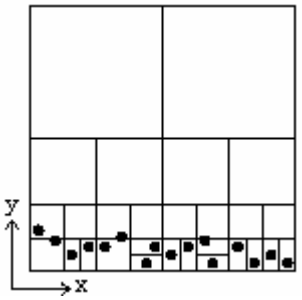

图 2-6

2.3.3 滑动中点分割策略

该策略起初的执行过程与“ 中点分割策略 ”是一致的，如果每次划定的分割超平面的左右两侧都有数据点，则利用此策略对点集进行划分将得到与利用“ 中点分割策略 ”对相同点集进行划分一样的结果。然面，当超平面的某一侧没有数据点，即有空叶节点产生时，我们将使超平面沿分割维滑动，直到它遇到第一个数据点为止。用更形式化的语言来说，如果分割维为第i维，并且所有的数据点在第i维上的坐标都大于（或小于）分割点坐标，我们将使超平面移动至第i维的最小（或最大）坐标处，也就是让超平面与第一个数据点相遇。设此点为p，则p点在第i维上的坐标会被确定为分割点，数据集将被划分为两部为：p点，p点以外的其它点。

显然本策略不会产生空叶节点，它在一定程度对“ 中点分割策略 ”进行了优化。不足在于，最终产生的超矩形仍可能俱有较高的长宽比。

图 2-7 示意了二维空间中分割超平面的划动，虚线表示按“ 中点分割策略 ”得到的超平面，箭头代表平面的滑动方向，实线代表平面最终所处的位置。

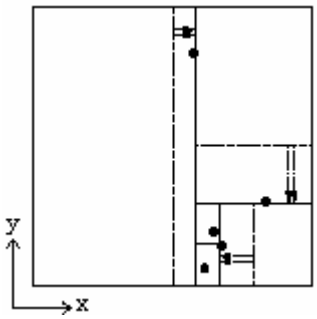

图 2-7

图2-8示意的是利用滑动中点分割策略对图2-4所示的数据点集进行分割所得到的kd树。

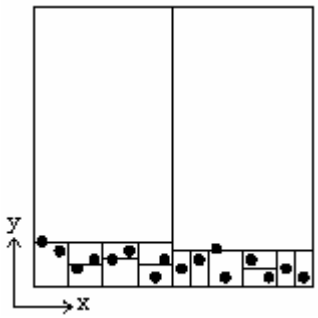

图2-8

2.3.4 分割策略的选择

根据Arya等人的理论我们知道：在kd树中，最终得到的超矩形的平均长宽比越小，空叶节点越少，则检对树进行检索进的效率越高[20]。

由2.3节中对三种常见分割策略的描述不难看出得出以下结论：

（1）对同一数据集进行划分，使用“ 滑动中点分割策略 ”产生的超矩形的平均长宽比通常不会超过使用“ 标准分割策略 ”产生的超矩形的平均长宽比。

（2）使用“ 滑动中点分割策略 ”不会产生空叶节点。从这点上说，“ 滑动中点分割策略 ”要优于“ 中点分割策略 ”。

可见，“ 滑动中点分割策略 ”是对“ 标准分割策略 ”和“ 中点分割策略 ”的一种权衡和优化，因此在实现 kd 树时，我们采用“ 滑动中点分割策略 ”。

2.4 本章小结

本章首先对 kd 树的概念进行了介绍，并以一个例子对其加以阐示。接着以 Bentley 的在[9]中提出的算法为例，说明了 kd 树的建立算法。然后我们又对 kd 树建立时的分割策略进行了较为详细的描述，最后对三种分割策略进行了比较。为下文基于 kd 树的异常检测系统的设计与实现打下了基础。

第3章 基于 kd 树的异常检测系统的设计与实现

3.1 系统总体框架

3.1.1 一个通用框架的简单介绍

作为完整的网络数据异常检测系统，一般包括如下几个功能模块[14]：捕包模块、数据预处理模块、数据分析模块以及报警模块。其中数据分析模块往往对异常检测系统检测性能优劣起到了决定性的作用。

一个较为通用的系统框架如图 3-1 所示。

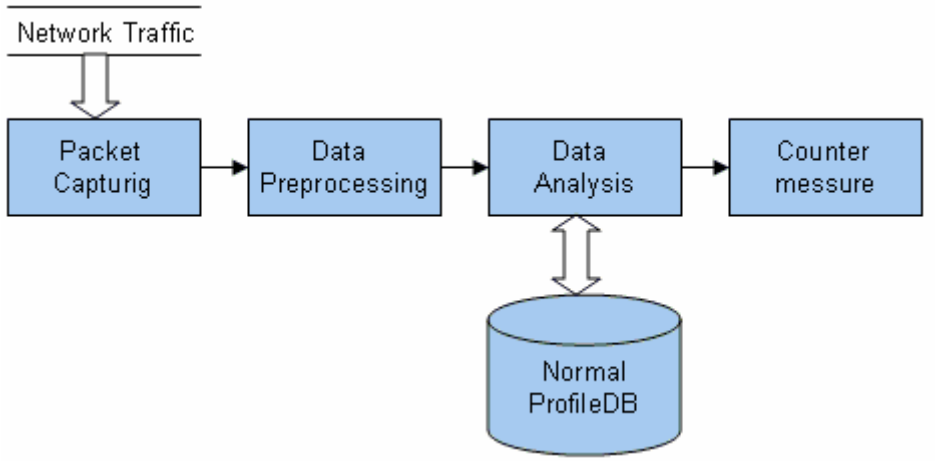

图 3-1

网络数据包的捕获是进行网络数据异常检测的第一步，接下来将捕获到的包进行数据的预处理，其中包括结合时间特征对相关数据包中内容的解析和提取，这一步骤结束之后就得到了数据分析模块可以处理的数据形式。数据分析模块根据已经建立的自组织特征向量模型，对提取出来的数据进行聚类分析，分析结果交送报警模块，对可能发生的网络攻击采取相应措施。

3.1.2 相关概念

在对系统的总体框架进行描述之前，先让我们定义如下几个概念：

(1)用户轮廓

把对目标系统活动的发起者成为主体（可以泛指计算机用户，也可以是主机或者程序进程）；系统管理的资源称为客体，那么用户轮廓（Profile）就是用来描述主体对客体行为特征（Behavior Character）的一种结构，而这种结构可以通过统计变量或者模型的形式表现出来。

Denning 于 1986 年给出了一个基于用户轮廓（profile）的入侵检测系统模型。基本思想是：通过对系统审计迹数据的分析建立起系统主体（单个用户、一组用户、主机甚至是系统中的某个关键的程序和文件等）的正常行为特征轮廓；检测时，如果系统中的审计迹数据与已建立的主体的正常行为特征有较大出入就认为是一个入侵行为。特征轮廓是借助主体登录的时刻、登录的位置、CPU 的使用时间以及文件的存取等属性，来描述它的正常行为特征。当主体的行为特征改变时，对应的特征轮廓也相应改变。

目前这类入侵检测系统多采用统计或者基于规则描述的方法建立系统主体的行为特征轮廓。

(2)对网络数据流的抽象

将网络数据流以矢量的形式表示为特征矢量 $X=(X_1,X_2,\cdots X_k)$，x_i 为每个网络数据流特征向量的分矢量；特征矢量 x 为 k 维特征空间 $\mathbf{R}^k$ 的一个点，$\mathbf{R}^k$ 为欧式空间。由此，我们将网络数据流抽象成为了欧式空间 $\mathbf{R}^k$ 的一个点。

(3)偏差的定义

由 3.1.3 知，将网络数据进行抽像后，待测数据可以看成是多维空间上的点，而用户轮廓则可看成是多维空间上的点集。因此，待测数据与用户轮廓的差最终可以归结到多维空间上两点间的距离。

我们采用欧氏距作为距离的度量，定义如下：

$$d(i,j)=\left\|x_i-y_j\right\|=\sqrt{\sum_{p=1}^{k}(x_{i,p}-y_{j,p})^2}$$

其中 $x_i=(x_{i,1},x_{i,2},....,x_{i,p})$、 $y_i=(y_{i,1},y_{i,2},....,y_{i,p})$，他们分别表示一个 p 维数据对象。

有了距离的概念，我们可以将偏差定义如下：设 X 为待测数据包对应的多维空间点，N 为用户轮廓中点的个数，D_i 为 X 与用户轮廓中每个点的距离 ($1 \le i \le N$)，且 {D1,D2 … Di} 为由小到大的递增序列，则 d=(D1+D2+…+Di)/i 代表待测数据包与用户轮廓的偏差，其中 1≤i≤N。

3.1.3 设计思路及总体框架

使用 kd 树，对大量正常的网络数据进行多维空间上的最优相似性划分，完成用户轮廓的建立。当时行检测时，我们考察当前数据包与用户轮廓的偏差程，如果偏差大与某个事先设定的阈值，则将其判定为攻击，否则认定其为正常数据。

依照上述思路，本文所实现的入侵检测系统框架如图 3-2 所示。严格来说，它并不是一个完整的网络异常检测系统，省去了捕包和报警模块。它的输入是已经捕获的数据包，输出是检测后的结果，即数据包是否为攻击包。

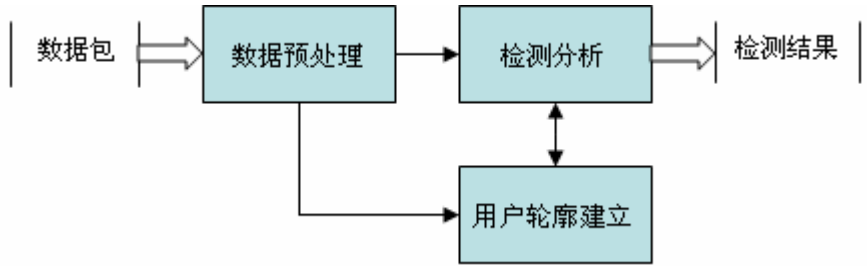

图 3-2

3.2 数据预处理模块

3.2.1 规格化

所谓规格化就是将有关属性数据按比例投射到特定小范围之中。之所以

要进行规格化，是为防止在距离计算时某些取值范围过大的分量掩盖了其它分量的作用，使得算出的距离不能完全体现两个向量间的差异。

3.2.2 最大最小规格化方法

该方法对被初始数据进行一种线性转换。设 min_A 和 max_A 属性 A 的最大和最小值。最大最小规格化方法将属性 A 的一个值 v 映射为 v’，且有 v’∈[new_min_A,new_max_A]，具体映射计算公式如下：

$$v'=(v-min_A)(new\_min_A-new\_max_A)/(max_A-min_A)+new\_min_A$$

最大最小规格化方法的好处是，保留了原来数据中存在的关系[3]。

3.2.3 模块设计

数据预处理模块的输入是原始数据包，输出是规格化后的经过特征提取的向量数据。

该模块包含两个子模块：规格化子模和特征提取子模块。其中，规格化子模块将原始数据抽象为多维空间中的向量点，并按照最大最小规格化方法对其进行规格化。特征提取子模块用于从规格化后的向量中提取特定的分量，组成用于检测的新向量。

3.3 用户轮廓建立模块

用户轮廓的建立过程，就是应用树形结构对多维空间数据点进行组织的过程，也就是 kd 树的建立过程。用户轮廓建立模块的输入是经过数据预处理模块抽象后的数据包，输出是指向 kd 树树根的指针。

3.3.1 主要的数据结构

3.3.1.1 最基本的数据类型

Typedef	double	ANNcoord	坐标
Typedef	double	ANNdist	距离
Typedef	double	ANNidx	k 邻近查询后的索引
Typedef	ANNcoord	*ANNpoint	数据点
Typedef	ANNpoint	*ANNpointArray	点集

Typedef ANNdist *ANNdistArray 待查点与其 k 个最近邻点的距离
Typedef ANNidxArray *ANNidxArray 索引序列的集合

3.3.1.2 节点

kd 树中的节点有两种类型：

（1）分裂节点：它主要记录裂信息，这些信息包括：分割维、分割点、当前超矩形的边界、指向左右子节点的指针。其类定义如下：

```
class ANNkd_split : public ANNkd_node      // splitting node of a kd-tree
{
    int                 cut_dim;     // dim orthogonal to cutting plane
    ANNcoord            cut_val;           // location of cutting plane
    ANNcoord   cd_bnds[2];  // lower and upper bounds of rectangle along cut_dim
    ANNkd_ptr  child[2];           // left and right children
public:
    ANNkd_split(                                    // constructor
        int cd,                                     // cutting dimension
        ANNcoord cv,                                // cutting value
        ANNcoord lv, ANNcoord hv,                   // low and high values
        ANNkd_ptr lc=NULL, ANNkd_ptr hc=NULL)       // children
        {
            cut_dim         = cd;           // cutting dimension
            cut_val         = cv;           // cutting value
            cd_bnds[ANN_LO] = lv;           // lower bound for rectangle
            cd_bnds[ANN_HI] = hv;           // upper bound for rectangle
            child[ANN_LO]   = lc;           // left child
            child[ANN_HI]   = hc;           // right child
        }
    ~ANNkd_split()                                  // destructor
}
```

（2）叶节点：它包含划分后的数据点的标号信息，叶节点通过维护一个数组存放这些数据点的标号。其类定义如下：

```
class ANNkd_leaf: public ANNkd_node        // leaf node for kd-tree
```

```
{
    int                 n_pts;      // number of points in bucket
    ANNidxArray         bkt;        // bucket of points
public:
    ANNkd_leaf(                     // constructor
        int             n,          // number of points
        ANNidxArray     b)          // bucket
        {
         n_pts= n;                  // number of points in bucket
         bkt = b;                   // the bucket
        }
    ~ANNkd_leaf() { }               // destructor
};
```

3.3.1.3 树

这个类的主要功能是提供建立 kd 树的成员函数。其类定义如下：

```
class ANNkd_tree: public ANNpointSet
{
protected:
    int                 dim;        // dimension of space
    int                 n_pts;      // number of points in tree
    int                 bkt_size;   // bucket size
    ANNpointArray       pts;        // the points
    ANNidxArray         pidx;       // point indices (to pts array)
    ANNkd_ptr           root;       // root of kd-tree
    ANNpoint  bnd_box_lo;           // bounding box low point
    ANNpoint  bnd_box_hi;           // bounding box high point

    void SkeletonTree(              // construct skeleton tree
        int     n,                  // number of points
        int     dd,                 // dimension
        int     bs,                 // bucket size
```

```
        ANNpointArray pa = NULL,        // point array (optional)
        ANNidxArray pi = NULL);         // point indices (optional)

public:
    ANNkd_tree(                                 // build skeleton tree
        int                 n = 0,              // number of points
        int                 dd = 0,             // dimension
        int                 bs = 1);    // bucket size
    ANNkd_tree(                                 // build from point array
        ANNpointArray  pa,              // point array
        int                 n,                  // number of points
        int                 dd,                 // dimension
        int                 bs = 1,             // bucket size
    ~ANNkd_tree();                      // tree destructor
};
```

3.3.2 kd 树建立流程

树的构建是基于建造 kd 树的一种标准算法的。这个过程是通过一种简单的分割～胜出策略执行的。这种策略不断使用超平面将点集进行分割。当子集中点的数目小于 bucket size 时，我们就把这些点放到一个叶节点的 bucket 中。整个过程涉及到 ANNkd_tree 类中的三个函数：ANNkd_tree( )、rkd_tree( )、SkeletonTree( )。

ANNkd_tree( )是类的构造函数，它调用其余两个函数完成树的建立，其流程见图 3-3。

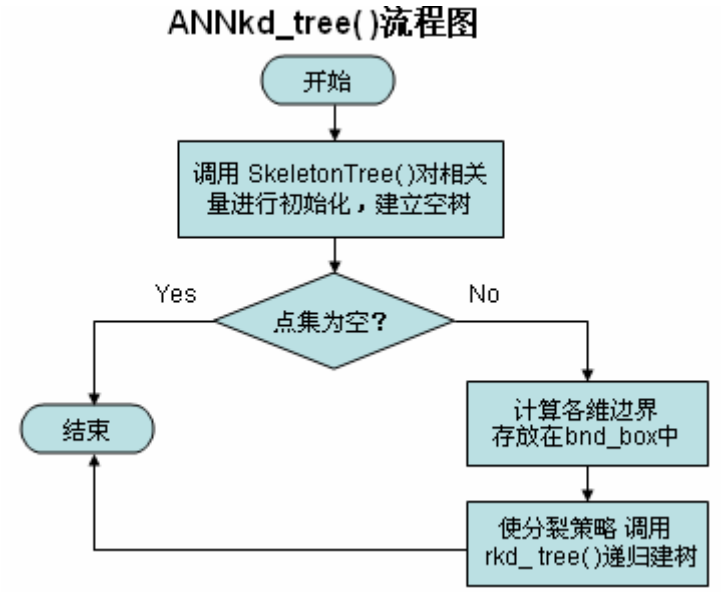

图 3-3

SkeletonTree( )主要完成建树前的一些初始化工作，其流程见图 3-4。

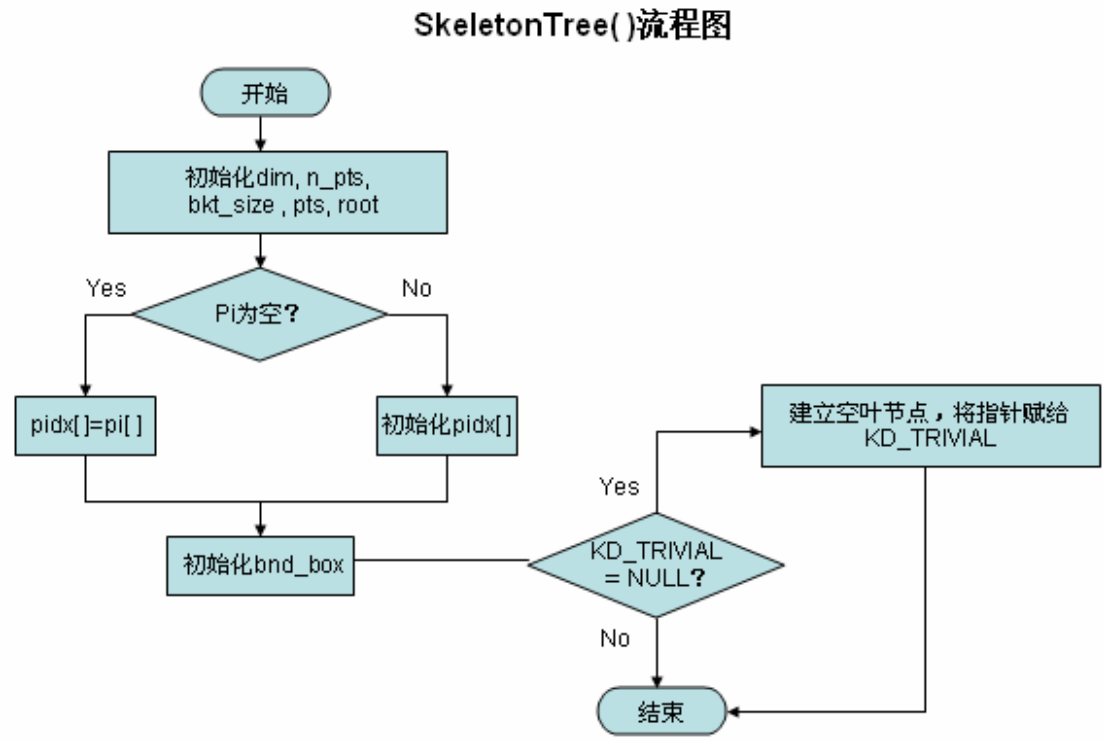

图 3-4

rkd_tree( )被用来进行递归建树，其流程见图 3-5

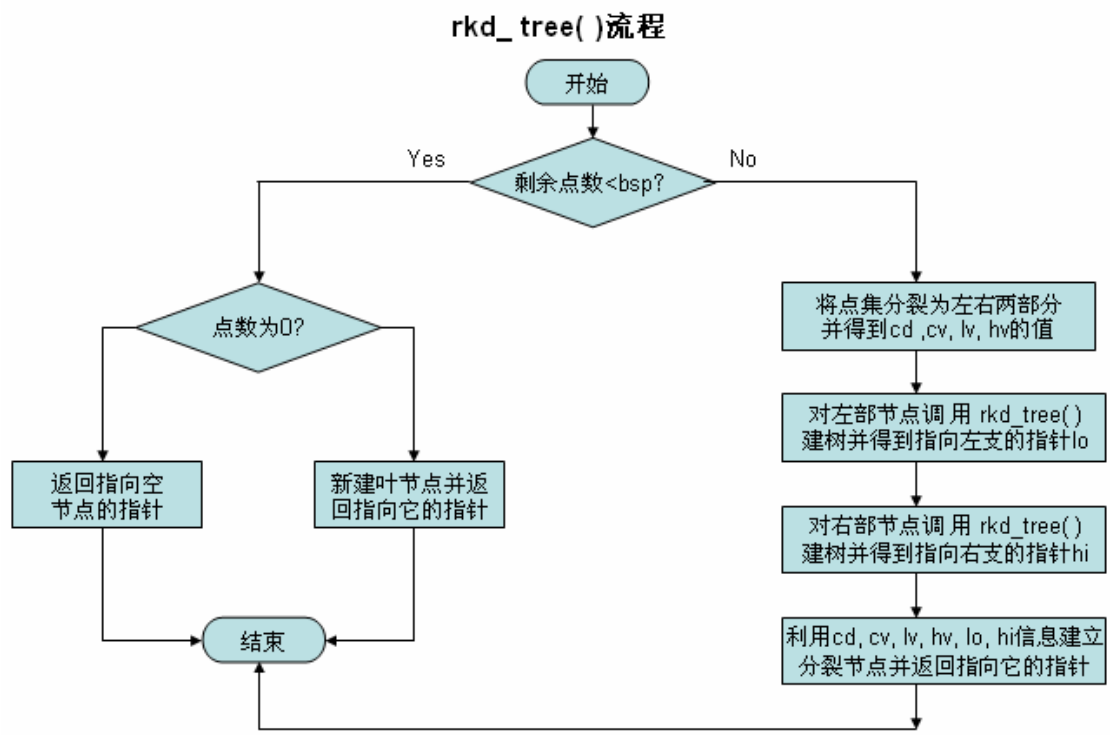

图 3-5

3.4 检测分析模块

检测分析模块的主要功能是判定待测点是否为攻击数据，它包含距离计算和统计分析两个子模块。的作用是计算待测点与轮廓的偏差，如果这个偏差大于事先设定的某个阈值，测将被测点视为攻击，否则认为其为正常数据。

3.4.1 距离计算子模块

该子模块的作用是计算待测数据点与用户轮廓的偏差。

3.4.1.1 数据点与节点间的距离

树中的每个节点对应一个超矩形，数据点与节点之间的距离也就是多

维空间中，点与超矩形之间的距离。

设 N（x_1,x_2，…，x_n）为 n 维空间中的点，R（B_1，B2，…，B_n）为多维空间中的超矩形。其中 Bi 为第 i 维上矩形的边界，B_i.h 表示上边界，B_i.l 表示下边界。我们给出如下定义：

（1）维距：在第 i 维上，如果 x_i>B_i.h 则维距为 x_i－B_i.h；如果 x_i<B_i.l 则维距为 B_i.l－x_i；如果 B_i.l≤xi≤B_i.h 则维距为 0。

（2）点形距：将各维上维距的平。方和定义为多维空间中点与超矩形之间的距离。

3.4.1.2 工作流程

模块首先在树中搜出与待测点距离最近的叶节点，然后计算叶节点中每个数据点与待测点的距离，跟据用户的要求选出与待测点距离最近的 i 个点，并按 3.1.2 的定义计算出待测点与轮廓的偏差。

在搜索时，我们采用的是 Firedman，Bentley 和 Finkel，在[10]中提出的算法。该算法是递归执行的，当遇到 kd 树的节点时，我们先访问距查询点最近的孩子节点，每次计算当前节点与待测点的距离并作累加。返回时，我们考察当前节点的另外一个孩子节点，如果待测点与另一孩子节点的距离小于当前的最小距离，则访问另外一个孩子节点，否则将其忽略。

图 3-6 为子模块输出的截图，其文件格式如下：待测点坐标，与之最近的轮廓上的点的序号，偏差。

3.4.2 统计分析子模块

该子模块主要是根跟设定好的阈值对距离计算子模块输出的文件进行分析。如果数据点对应的偏差大于阈值则将其定为攻击，否则视其为正常。除此之外，该模块还能以对模形的误报漏报率进行统计，以便我们对模型和阈值的设定进行分析评价。

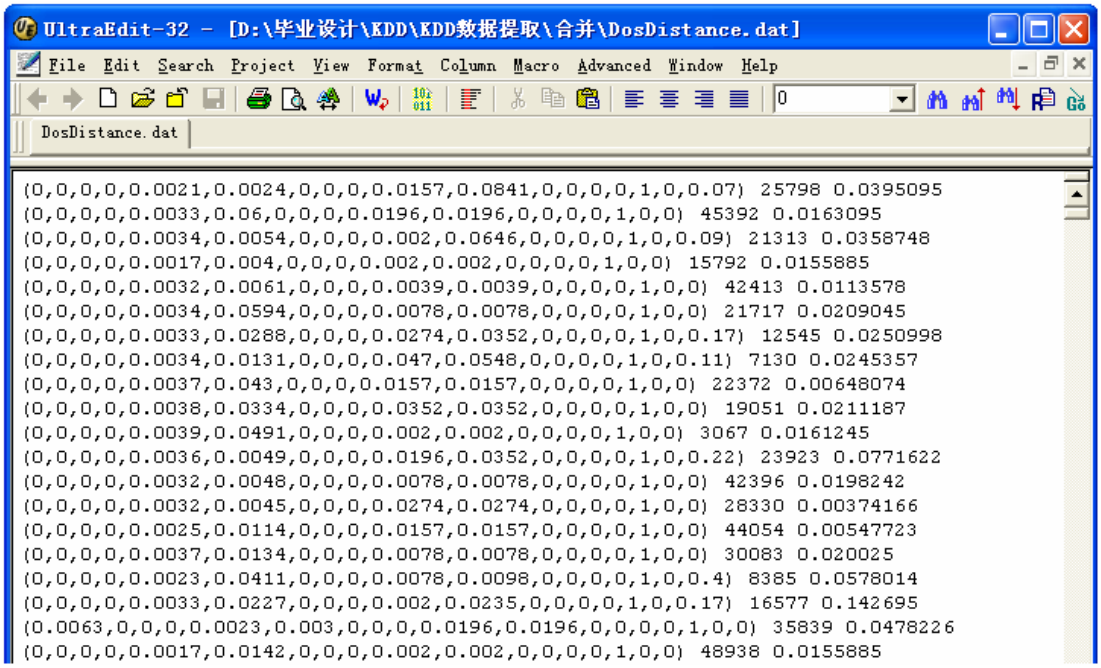

图 3-6

3.5 本章小节

本章中完整介绍了基于 kd 树的异常检测系统的总体设计及实现。首先，我们给出了用户轮廓、偏差等必要的概，在此基础上对总体设计思路及系统的框架进行了描述。接着，以用户轮廓建立模块为重点，对系统各模块的功能、实现及工作流程作了阐述。至此，一个基于 kd 树的异常检测系统已经构架完成。

第4章 实验

4.1 实验数据与实验环境介绍

4.1.1 KDD CUP 1999 数据集简介

本章的所有试验都应用于KDD Cup 1999 Data [KDD99]上。该数据集首先在与KDD99 同时举办的第三届国际知识发现和数据挖掘工具竞赛上使用，它包含了在军事网络环境中仿真的各种入侵。检测网络入侵的软件能使网络免遭各种非授权用户及潜在入侵者的侵扰。这次竞赛的任务就是构造一个网络入侵检测器，即一个可以区分坏连接（入侵或攻击）和好的正常连接的预测模型。该数据集提供了从一个模拟的美国空军局域网上采集来的9个星期的网络连接数据，其训练数据集包含5 百万个连接数据，测试数据集包含了2百万个连接数据。一个连接就是在规定的协议下、在规定的时间内完成的起始并终止的TCP 分组序列，这些序列在固定的源IP 地址与目的IP 地址之间进行数据传输。每个连接都带一个类标识，或者是正常，或者是某个具体的攻击类型。每个连接记录大概有100 个字节。图4-1 给出了该数据集所含的数据记录示例。

0,udp,private,SF,105,147,0,0, …,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,snmpgetattack
0,udp,private,SF,105,147,0,0, …,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00, snmpgetattack
282,tcp,ftp,SF,162,597,0,0, …,0,0,1,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00, warezmaster
0,tcp,ftp_data,SF,12,0,0,0, …,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00, warezmaster
0,tcp,http,SF,282,24572,0,0,…,0,0,3,3,0.33,0.33,0.00,0.00,1.00,0.00,0.00,normal
0,udp,private,SF,105,147,0,0, …,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,normal
0,tcp,pop_3,S0,0,0,0,0,…,0,0,1,2,1.00,1.00,0.00,0.00,1.00,0.00,mscan
0,tcp,telnet,S0,0,0,0,0,…,0,0,1,12,1.00,0.25,0.00,0.58,1.00,0.00,mscan
0,tcp,private,REJ,0,0,0,0,…,0,0,142,9,0.00,0.00,1.00,1.00,0.06,0.06,0.00,neptune

图 4-1

每条连接包括42个字段，每字段含义如表4-1所示,其中C表示特征取值为连续值，D表示特征取值为离散取值。

表4-1

标号	特征	意义描述	类型
1	duration	length (number of seconds) of the connection	C
2	protocol_type	type of the protocol, e.g. tcp, udp, etc.	D
3	service	network service on the destination, e.g., http, telnet,etc.	D
4	src_bytes	number of data bytes from source to destination	C
5	dst_bytes	number of data bytes from destination to source	C
6	flag	normal or error status of the connection	D
7	land	1 if connection is from/to the same host/port;0 otherwise	D
8	wrong_fragmen	number of ``wrong'' fragments	C
9	urgent	number of urgent packets	C
10	hot	number of ``hot'' indicators	C
11	num_failed_logins	number of failed login attempts	C
12	logged_in	1 if successfully logged in; 0 otherwise	D
13	num_compromised	number of ``compromised'' conditions	C
14	root_shell	1 if root shell is obtained; 0 otherwise	D
15	su_attempted	1 if ``su root'' command attempted; 0 otherwise	D
16	num_root	number of ``root'' accesses	C
17	num_file_creations	number of file creation operations	C
18	num_shells	number of shell prompts	C
19	num_access_files	number of operations on access control files	C
20	num_outbound_cmd	number of outbound commands in an ftp session	C
21	is_hot_logi	1 if the login belongs to the ``hot'' list; 0 otherwise	D
22	is_guest_login	1 if the login is a ``guest''login; 0 otherwise	D
23	count	number of connections to the same host as the current connection in the past two seconds	C
24	srv_count	number of connections to the same service as the current connection in the past two seconds	C

25	serror_rate	% of connections that have ``SYN'' errors	C
26	srv_serror_rate	% of connections that have ``SYN'' errors refer to same-service connections	C
27	rerror_rate	% of connections that have ``REJ'' errors	C
28	srv_rerror_rate	% of connections that have ``REJ'' errors refer to same-service connections	C
29	same_srv_rate	% of connections to the same service	C
30	diff_srv_rate	% of connections to different services	C
31	srv_diff_host_rate	% of connections to different hosts	C
32	dest_host_count	number of connections to the same destination host as the current connection in the past two seconds	C
33	dest_host_srv_count	number of connections to the same service as the current connection in the past two seconds for the destination host	C
34	dest_host_same_srv_rate	%of the destination host connections to the same service	C
35	dest_host_diff_srv_rate	% of the destination host connections to different services	C
36	dest_host_same_srv_ port_rate	% of the destination host connections to the same services and ports	C
37	dest_host_srv_diff_host_rate	% of the destination host connections to the same services but different host.	C
38	dest_host_serror_rate	% of the destination host connections that have ``SYN'' errors	C
39	dest_host_srv_serror_rate	% of connections that have ``SYN'' errors refer to same-service connections	C
40	dest_host_rerror_rate	% of the destination host connections that have ``REJ'' errors	C
41	dest_host_srv_rerror_rate	% of connections that have ``REJ'' errors refer to same-service connections	C
42	Label	the connection is normal or attack	D

4.1.2 数据集所含攻击类型

数据集中出现的攻击有以下四种类型：

DOS：拒绝服务攻击，如ping of death, teardrop, smurf,syn flood；

R2L：来自远程机器的非法访问，如guessing password；

U2R：普通用户对本地超级用户特权的非法访问，如各种buffer overflow攻

击；

Probing：监视和其它探测活动，如 port scanning，ping sweep。

4.1.2 实验环境

PC 机一台，配置如下：

处理器：Pentium4 2.4GHz　　内存：256MB

操作系统：Linux9.0，WindowXP。

4.2 Wenke Lee 的检测属性集理论

观察 KDD 原始数据集，可以发现每类攻击包括的攻击种类很多，如 Dos 类攻击就包括 land,neptune,teardrop 等。种攻击，每种攻击表现出来的连接特性并不完全相同，但是有很多共性，Wenke Lee 等人通过数据挖掘的办法发现了这种共性，并且发现检测不同的攻击需要采用不同的属性集合来检测才比较有效[21]。

我们将表4-1中的41个特征（label除外）划分为如下4个检测属性集。

(1)基本属性集：

特征1～9，包括连接的持续时间、协议、服务、发送字节数等；

(2)内容属性集：

特征12～22，即用领域知识获得与信包内容相关的属性，如：连接中的hot标志的个数、本连接中失败登陆次数、是否成功登陆等；

(3)流量属性集：

特征22～31，即基于时间的与网络流量相关的属性，这类属性又分为两种集合，一种为Same Host属性集，即在过去2秒内与当前连接具有相同目标主机的连接中，有关协议行为、服务等的一些统计信息，另外一种为Same Service属性集，即在过去2秒内与当前连接具有相同服务的连接中作出的一些统计信息。

(4)主机流量属性集：

特征 32～41，即基于主机的与网络流量相关的属性，这类属性是为了发现慢速扫描而设的属性，获取的办法是统计　在过去的 100 个连接中的一些统计特性，如过去 100 个连接中与当前连接具有相同目的主机的连接数、

与当前连接具有相同服务的连接所占百分比等。

表 4-2 给出了检测不同类的攻击需要的检测属性集合。

表 4-2

类别	用于检测本类攻击的属性集
DoS	基本属性集 + 流量属性集
Probe	基本属性集 + 流量属性集 + 主机流量属性集
U2R	基本属性集 + 内容属性集
R2L	基本属性集 + 内容属性集

4.3 ROC 曲线

在介绍 ROC 曲线[13]之前，我们先来明确两个概念：

（1）检测率（DR）：指被监控系统在受到入侵攻击时，检测系统能够正确报警的概率。

计算公式为：DR=被检测出来的异常样本数据/异常样本总数。

（2）误报率（FPR）：指检测系统在检测时出现误报的概率。

计算公式为：FPR=正常样本被误报为异常的样本数/正常样本数。

实际的 IDS 的实现总是在检测率和误报率之间徘徊，检测率高了，误报率就会提高;同样误报率降低了，检测率就会降低。一般地，IDS 产品会在两者中取一个折衷，并且能够进行调整，以适应不同的网络环境。美国的林肯实验室用接收器特性(ROC，Receiver Operating Characteristic)曲线来描述 IDS 的性能。该曲线准确刻画了 IDS 的检测率与误报率之间的变化关系。ROC 广泛用于输入不确定的系统的评估。根据一个 IDS 在不同的条件(在允许范围内变化的阈值，例如异常检测系统的报警阈值等参数)下的率和检测率，分别把误率和检测率作为横坐标和纵坐标，就可做出对应于该 IDS 的 ROC 曲线。ROC 曲线与 IDS 的报警阈值具有对应的关系。

通常，一条 ROC 曲线的大致形态如图 4-2 所示。一般我们将阈值设在曲的拐点附近（图中的 1 处），认为此处是漏报与检测率的最佳平衡点。

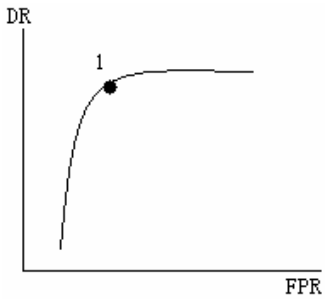

图 4.2

4.4 实验数据准备

原数据集过于庞大，我们选取其中较有代表性的 10%作为实验数据，即 kdd 原始数据包中的 kddcup.data_10_percent.gz 文件。对文件中的数据我们作如下处理：

（1） 将正常数据与攻击数据分离。

（2） 将攻击数据分离为三个集合：Dos，Probe，U2R－R2L。之所以将 U2R 和 R2L 是因检测它们时用到的属性集是相同的。

（3） 构造正常流量集。从（1）中正常数据中抽取一部分正常连接信息数据，使其包含各种协议（TCP,UCP,ICMP）及服务（ftp，telnet 等），形成正常流量集，用于对模型进行训练，形成用户轮廓。

（4） 构造测试数据集。对（2）中分离出的的三个攻击集合分别按照 1:1，1:1，1:9（以上比例为攻击：正常）混入正常连接数据。这样作的目的是为了使测试数据整接近于攻击发生时的真实网络流量配比[1]。形成的测试数据集分别称为 Dos，Probe，U2R－R2L。

4.5 实验流程及结果

4.5.1 实验流程

整个实验过程包括数据预处理，轮廓建立，攻击检测，阈值调整，检测率、误报率统计等多个步骤。图 4-3 以对 Dos 攻击的检测为例给出了实验的

数据处理流程图。其中圆角矩形框代表数据处理过程，直角矩形框代表每步处理后得到的文件。

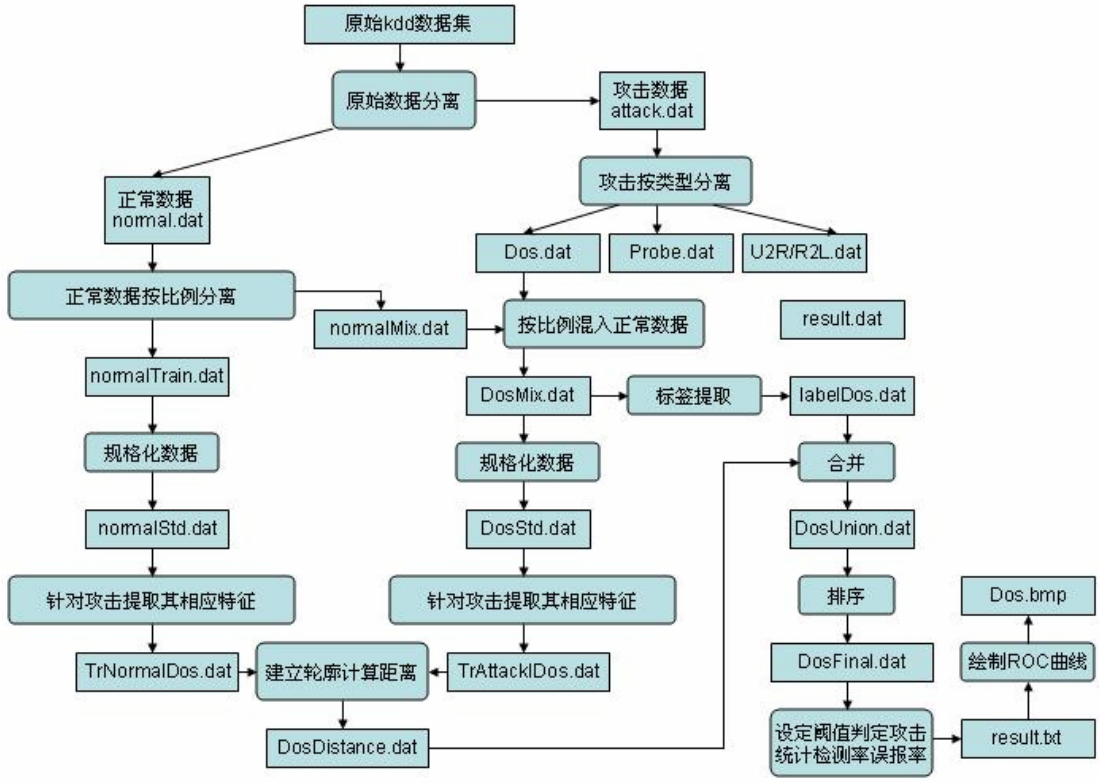

图 4-3

几点说明：

(1)规格化数据：即按照 3.2.2 所述方法对实验数据进行规格化。

(2)针对攻击提取相应特征：就是按照表 4-2，针对欲检测的攻击类型提取数取数据流中的不同字段。

(3)按比例混入正常数据：按照 4.4 中所述的攻击与正常数据的配比将数据混合，形成测试数据集。

(4)标签提取、合并：在计算偏差前将测试数据集中各数据点的 label 属性分离出来，计算偏差后再将其赋回给相应的数据点，最后按照偏差由小到大的顺序数据点排序，其目是便于以后对阈值的设置和对检测率和误报率的统计。

图 4-4 为排序后得到的文件的截图，文件格为：待测点坐标，与之最近

的轮廓上的点的序号、偏差、标签(其中 1 表示正常数据，－1 表示攻击数据)。

```
(0,0,1,0,0.0989,0,0,0,0,0.0215,0.0215,0,0,0,0,1,0,0)  47641  0  1
(0,0,0.0833,0,0.011,0.0033,0,0,0,0.002,0.0059,0,0,0,0,1,0,1)  47613  0.0114018  1
(0,0,0.1667,0,0.0001,0.0017,0,0,0,0.002,0.002,0,0,0,0,1,0,0)  47517  0.0155563  1
(0,0,0.0833,0,0.0187,0.0034,0,0,0,0.002,0.002,0,0,0,0,1,0,0)  28034  0.0155885  1
(0,0,0.0833,0,0.0168,0.0034,0,0,0,0.002,0.0039,0,0,0,0,1,0,1)  30798  0.0255147  1
(0.0001,0,0.0833,0,0.0293,0.0034,0,0,0,0.002,0.0039,0,0,0,0,1,0,1)  43343  0.0256969 1
(0,0,0.3333,0.2,0,0,0,0,0,0.0313,0.0294,0.94,1,0,0,0.94,0.12,0)  40441  2.18419  -1
(0,0,0.3333,0.2,0,0,0,0,0,0.0117,0.0098,0.83,1,0,0,0.83,0.33,0)  40441  4.39506  -1
(0,1,0.75,0,0.0117,0,0,0,0,0.6184,0.6184,0,0,0,0,1,0,0)  23352  7.07206  -1
(0,1,0.75,0,0.0117,0,0,0,0,0.9961,0.9961,0,0,0,0,1,0,0)  31054  7.07252  -1
(0,1,0.75,0,0.0117,0,0,0,0,0.998,0.998,0,0,0,0,1,0,0)  34054  7.07263  -1
(0,1,0.75,0,0.0117,0,0,0,0,1,1,0,0,0,0,1,0,0)  37064  7.07287  -1
(0,1,0.75,0,0.0117,0,0,0,0,1,1,0,0,0,0,1,0,0)  17052  7.17086  -1
(0,1,0.75,0,0.0117,0,0,0,0,1,1,0,0,0,0,1,0,0)  27076  7.19284 -1
(0,1,0.75,0,0.0117,0,0,0,0,1,1,0,0,0,0,1,0,0)  30084  7.60187 -1
```

图 4-4

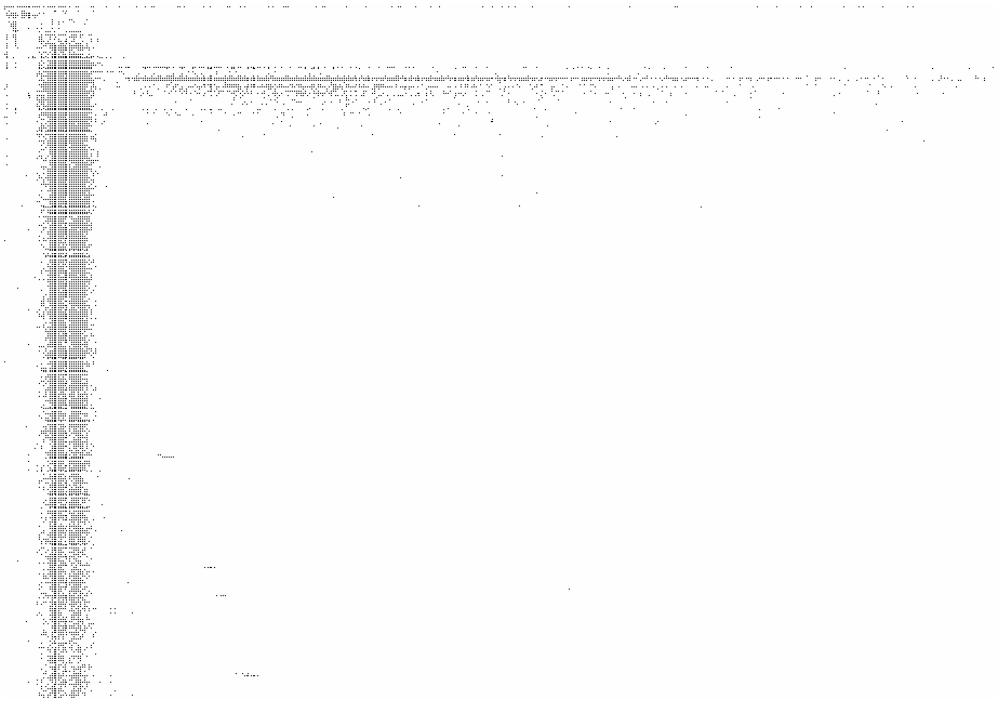

图 4-5

4.5.2 实验结果

针对不同的攻击，我们从正常流量集中提取不同的属性集形成新的多维

向量集合，对模型进行训练，即建立 kd 树。图 4-5 针对 Dos 攻击形成的多维向量集在空间中的分布的展示。图 4-6 使用该数据集建立的 kd 在二维空间上的展示，由于版面所限，该图展示的不是一棵完整的树，而是树的一个局部，图中的数据点对应的是图 4-7 右上方的一个局部的数据。

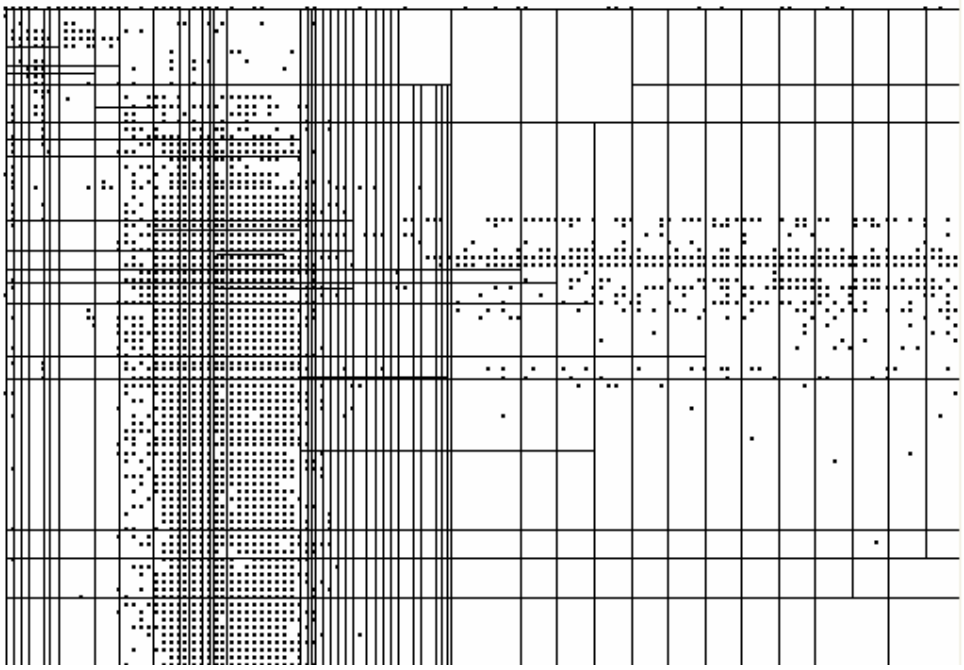

图 4-6

建好模型后，我们要作的就是让模型对测试数据进行检测。设待测点距模型上最近点的距离为 D1，次近点的距离为 D2，第三邻近点距离为 D3。

（1） 使用 D1 做为测点与模型的偏差度量，针对不同的测试数据集我们得到的 ROC 曲线如下：

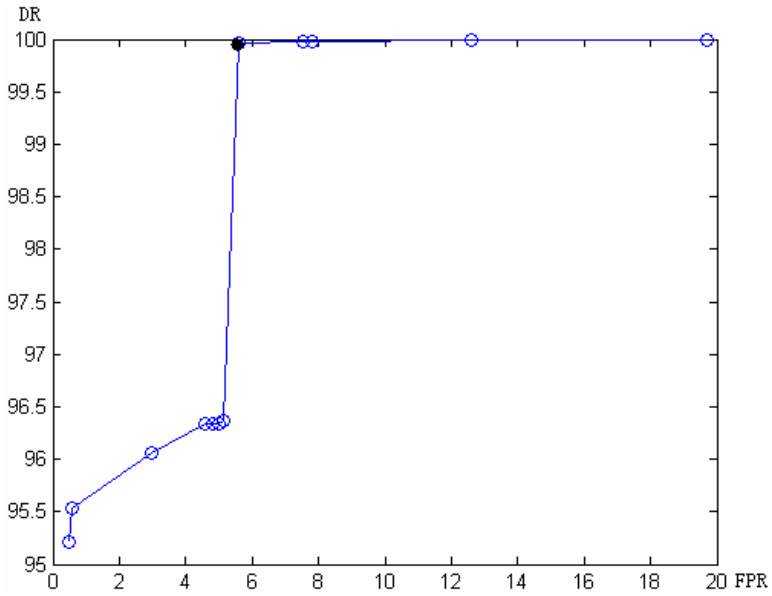

图 4-7 对含 Dos 攻击数据检测所得的 ROC 曲线

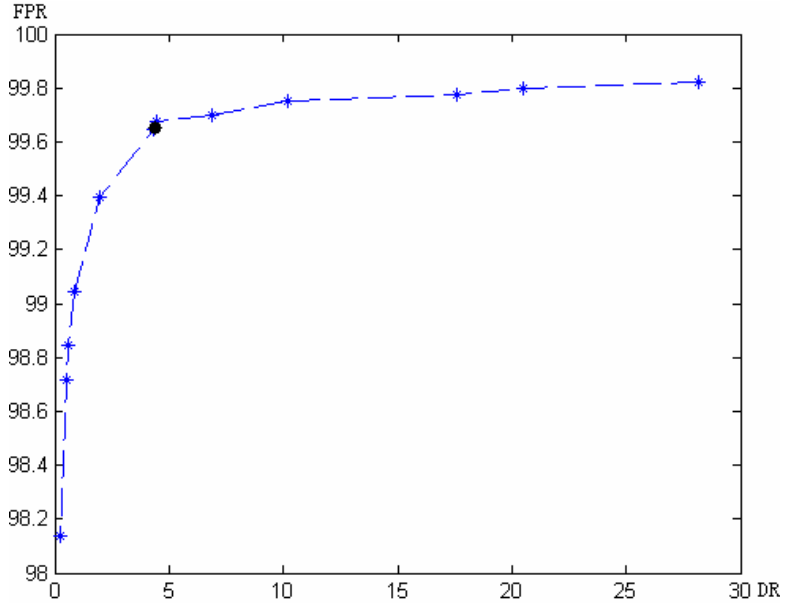

图 4-8 对含 Probe 攻击数据检测所得的 ROC 曲线

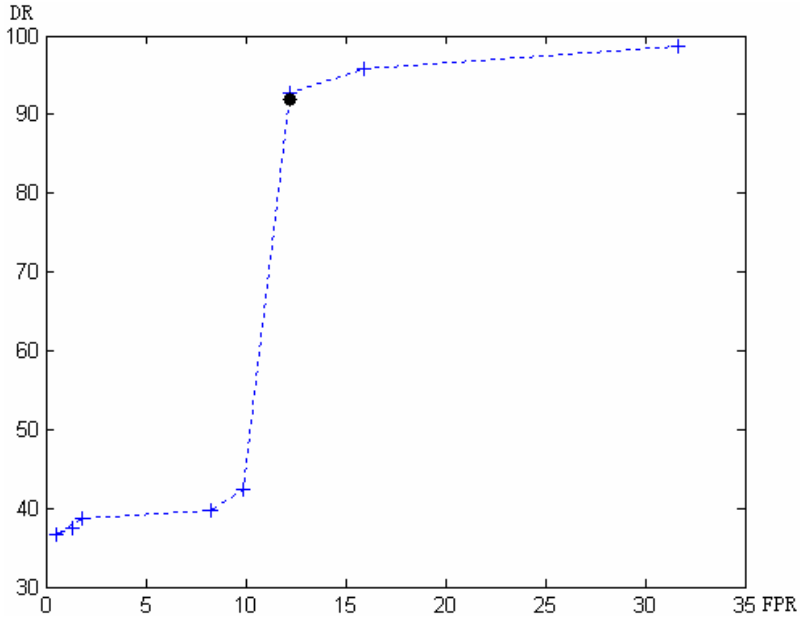

图 4-9 对含 U2R－R2L 攻击数据检测所得的 ROC 曲线

（2）使用（D1＋D2）/2 做为测点与模型的偏差度量，针对不同的测试数据集我们得到的 ROC 曲线如下：

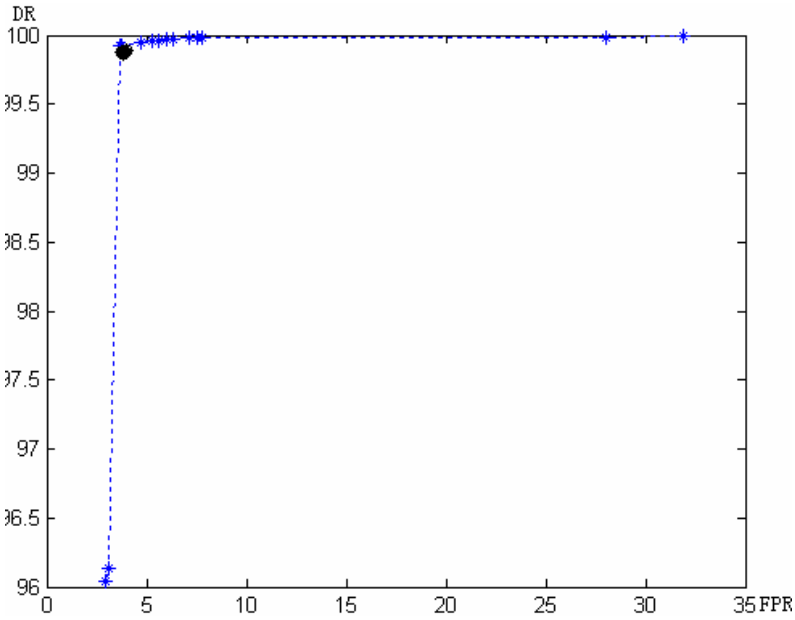

图 4-10 对含 Dos 攻击数据检测所得的 ROC 曲线

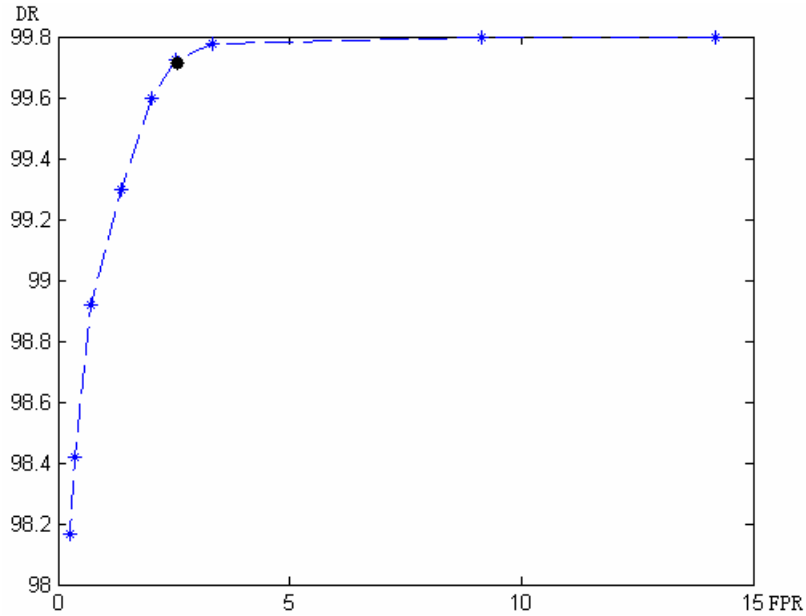

图 4-11 对含 Probe 攻击数据检测所得的 ROC 曲线

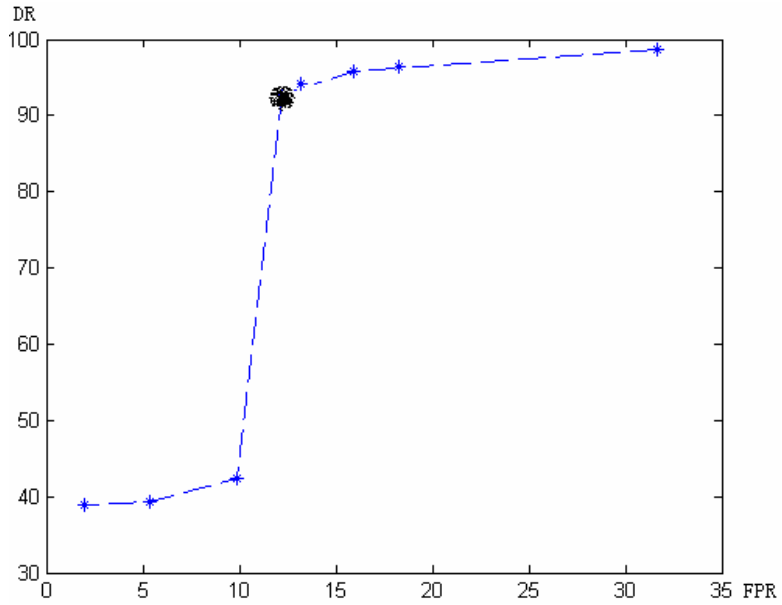

图 4-12 对含 U2R－R2L 攻击数据检测所得的 ROC 曲线

（3）使用（D1＋D2＋D3）/3 做为待测点与模型的偏差度量，针对不同的测试数据集我们得到的 ROC 曲线如下：

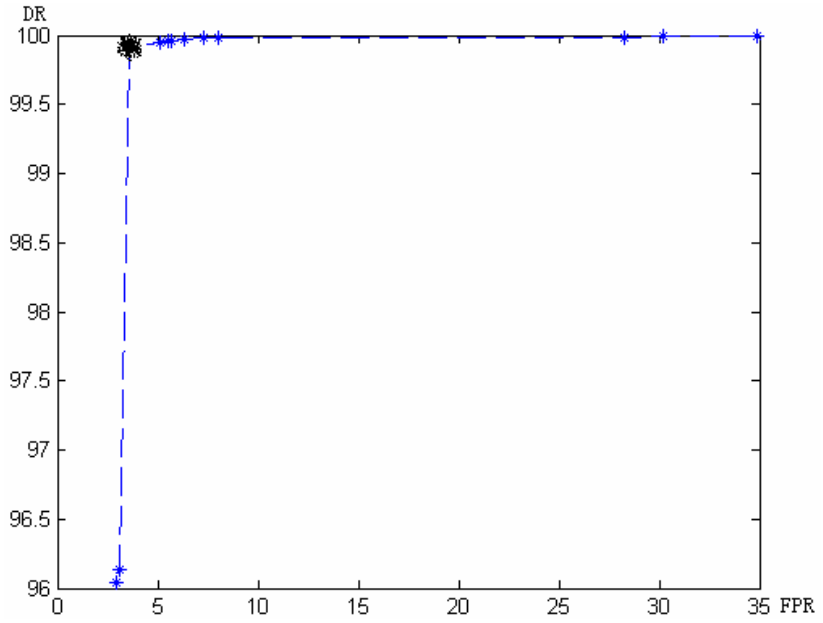

图 4-13 对含 Dos 攻击数据检测所得的 ROC 曲线

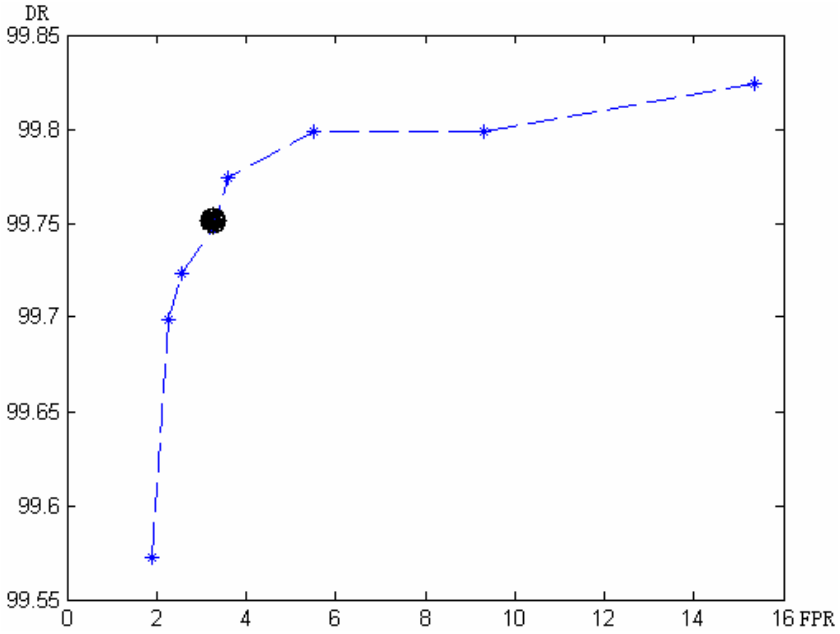

图 4-14 对含 Probe 攻击数据检测所得的 ROC 曲线

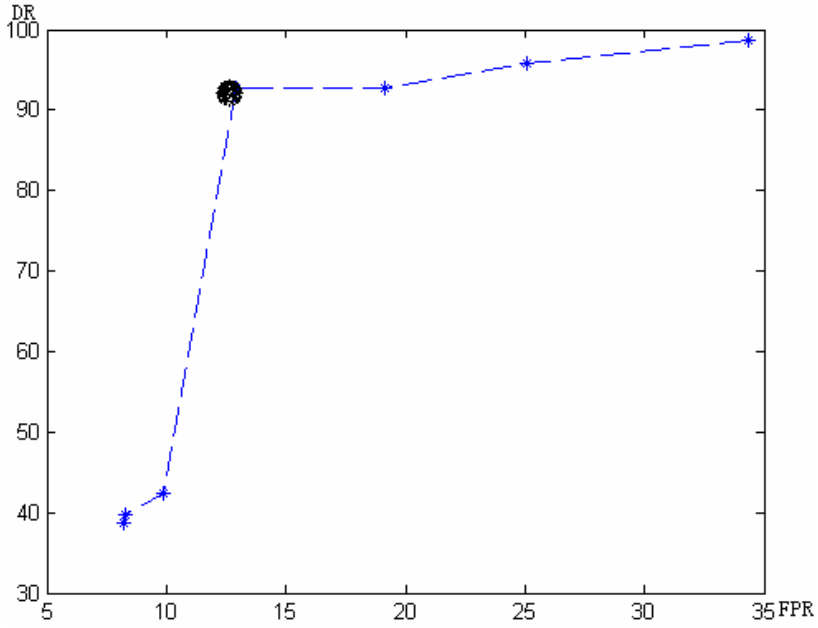

图 4-15 对含 U2R－R2L 攻击数据检测所得的 ROC 曲线

拐点处(在图中用黑色圆点标识)相应的误报率和检测率如表 4-3 所示：

表 4-3

	误报率%			检测率%		
	D1	(D1+D2)/2	(D1+D2+D3)/3	D1	(D1+D2)/2	(D1+D2+D3)/3
Dos	5.61385	4.70357	4.45173	99.9658	99.9421	99.9395
Probe	4.44889	3.34916	3.29918	99.6731	99.7737	99.7737
U2R/R2L	12.2538	12.2538	12.9249	92.7523	92.7523	92.7523

由表 4-3 易知，当使用(D1+D2)/2 作为偏差度量时，系统对 Dos 和 Probe 的检测性能有较大提高～在检测率基本不变的情况下误报率降低在 1%左右，对 U2R/R2L 的检测性能未发生改变；当使用(D1+D2+D3)/3 作为偏差度量时，系统对 Dos 和 Probe 的检测性能虽然也的提高，但提高的程度很小～在检测率基本不变的情况下误报率降低在 0.2%左右，对 U2R/R2L 的检测性能不升反降。

可见，在使用(D1+D2+D3)/3 作为偏差度量时，系统性检测性能提高十分有限，只是徒使计算量增加～对每个待测量要多算一个 D3。因此我们得出，将(D1+D2)/2 作为偏差度量是较为合理的。

4.6 本章小节

本章首先对 KDD 数据集进行了简要介绍，给出了数据集每条连接中各字段的含意。在此基础上我们给出了检测属性集，并按 Wenke Lee 的理论给出了针对每类不同的攻击所用到的检测属性集。接下去，我们对 IDS 系统的性能评测曲线～ROC 曲线的意义进行了解释。最后，我们给出了实验流程及实验结果。

结 论

本文的目标是实现基于 kd 树的网络异常检测，以实现对网络业务量数据中攻击的高效检测和抵御。现将来文的成果及目标完成情况总结如下：

(1) 对 kd 树算法的深入研究

通过对 kd 树算法的深入学习和研究，从理论上证明了将 kd 树用于异常检测领域的可行性。

(2) 对基于 kd 树的异常检测系统的模块化设计和实现

本异常检测系统采用模块化设计，使其能够添加其他异常检测训练算法并进行测试。这一点满足了研究的动态性和灵活性。

(3) 对基于 kd 树算法的网络异常检测模型的训练与测试

在 KDD 数据集的基础上形成训练数据集与测试数据集。使用训练数据集对系统进行训练，用试测数据集模拟网络流量对系统的性能进行测试。统计结果表明，系统对攻击（特别是 Dos 与 Probe 攻击）有较好的检测效果。

(4)对偏差度量的选择

由 4.4.2 的实验结果比较可知，采用最邻近与次邻近点和的均值作为待测数据与用户轮廓的偏差度量会使基于 kd 树的异常检测系统的性能达到最佳。

(5)对检测属性集的验证

Wenke Lee 等人提出检测不同的攻击需要采用不同的属性集合来检测才比较有效。本文采用其方法对不同类攻击进行检测，收到了较好的检测效果。

上述完成的工作中还存在一些局限性，需要进一步的工作来完善。

一方面，对系统对 U2R-R2L 攻击的检测效果不是特别另人满意。是否可以通过改进算法或者加入对报文内容的检测来提高对此类攻击的检测效果有待进一步研究。

另一方面，因为 KDD 数据具有模拟、离线的性质，所以在真实网络环境下的实时的异常检测结果尚待研究及验证。

参考文献

1 李辉 管晓宏 咎鑫 韩崇昭.基于支持向量机的网络入侵检测．计算机研究与发展 Vol.40,No.6 June 2003.

2 梁铁柱 入侵检测中的数据挖掘方法研究 中国人民解放军理工大学 博士论文 中图分类号：TP311

3 朱明 《数据挖掘》 清华大学出版社。

4 王晓程, 刘恩德, 谢小权. 攻击分类研究与分布式网络入侵检测系统. 计算机研究与发展. Vol.38, No.6: 727-734, 2002.6

5 W Richard Steven. TCP/IP 详解卷 1：协议[M].机械工业出版社, 2001-08

6 蒋建春，马恒太，任党恩，卿斯汉.网络安全入侵检测研究综述。软件学报. 2000.11：1461-1466.

7 孙总参，陶兰，齐建东，王保迎. 基于 kd 树的 k-means 聚类方法. 计算机工程与设计 Vol.25 2004.11

8 K.Sequeria,M.Zaki,ADMIT:Anomaly-base Data Mining for Intrusions, Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,Edmonton,July 2002.

9 Jon Louis Bentley Multidimensional Binar Search Trees Used for Associative Searching,ACM Student Award 1975**.**

10 Songrit Maneewongvatana and David M. Mount Analysis of Approximate Nearest Neighbor Searching with Clustered Point Sets. DIMACS Series in D Mathematics and Theoretical Computer Science

11 D.-Y.Yeung and C.Chow. Parzen-window netowrk intrusion detectors.In Proc.of the 16th Int'l Conf. On Pattern Recognition,volume 4,2002:385-388.

12 Wei Fan,et al.Using artificial anomalies to detect unknown and known network intrusions.In ICDM.2001:123-130.

13 王自亮 罗守山 杨义先. IDS 入侵检测系统的测试与评估 通信技术政策研究

14 韩东海 王超 李群 《入侵检测系统实例剖析》 清华大学出版社

15 Stephen Northcutt 等《入侵特征分析》 林琪 译 中国电力出版社

16 王勇 杨辉华 王行愚 何倩. 基于最小二乘支持向量机的 Linux 主机入侵检测系统. 计算机工程与应用 2005.2

17 李卓桓 《linux 网络编程》 机械工业出版社

18 Portnoy L, Eskin E, Stolfo S. Intrusion detection with unlabeled data using clustering. In: Barbara. Ded Proceeding of ACM CSS Workshop on Data Mining Applied to Security. Philadelphia: ACM Press. 2001.

19 Aleksandar Lazarevic, Aysel Ozgur, Levent Ertoz. A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. Department of Computer Science Department, University of Minnesota.

20 David M.Mount. ANN Programming Manual. Department of Computer Science and Institute for Advanced Computer Studies University of Maryland,College Park,Maryland

21 Wenke Lee,S J Stolfo,K W Mok. A data mining framework for building intrusion detection models. The 1999 IEEE Symposium on Security and Privacy,Oakland,CA,1999.

22 Kenji Yamanishi,et al.Online unsupervised outlier detection using finite mixtures with discounting learning algorithms.In Knowledge Discovery and Data Mining,2000:320-324.

23 A.Ghosh,A.Schwartzbard,A Study in Using Neural Networks for Anomaly and Misuse Detection,Proceeding of the 8th USENIX Security Symposium,1999.

24 M.Mahoney,P.Chan,Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks,Proceeding of the Eighth ACM SKGKDD Inernational Conference on Knowledge Discovery and Data Mining., Canana, Edmonton, July 2002:376-385.

25 H.Wang D.Zhang,and K.G..Shin.Detecting syn flooding attacks.In Proceedings of IEEE INFOCOM 2002,New York City,NY,June 2002.

附 录 1

对聚类点集近似最邻近搜索的分析

Songrit Maneewongvatana 和 David M.Mount

摘要

最邻近搜索是最基本的计算问题。给定 D 维空间中的一个点集，它包含 n 个数据点，最邻近搜索要解决的问题是用一个数据结构对这些点进行处理，以便给出一个查询点时，我们可以高效地在数据集中找到与该点距离最近的点。因为数据集可能很大，所以我们更关注数据结构所占的存储空间，希望将其控制在 O(dn)。

一种较为流行的用于最邻近搜索的数据结构是 kd 树及其变体，这项技术的基本思想是将分级的分解空间放进一个盒子中。在建立这种数据结构时的一个重要的问题是分割算法的选择，分割算法决定了分割维和用于空间分割的超平面。分割算法的选择在很大程度上决定了数据结构的效率，当数据点和查询点都聚集在低维子空间时，这点体现的尤为明显。这是因为较高的聚集性会让子划分生成长宽比极高的盒子。

我们将两种可选的分割策略与众所周知的优化 kd 树策略进行了对比。第一种叫做滑动中点分割策略。它试图寻求一种平衡，以使得子划分产生的盒子有一个长宽比的上限，且每个盒子中都含有数据点。第二种被称为最小不确定性分割策略，它是一种基于询问的搜索方式。这种策略中，我们除了要给出数据点集外，还要给出一个训练查询集，用于预处理。策略中使用了一种简单的贪心算法，以期选择的分割平面能在对训练集查找最邻近点时使得平均不确定性的总和最小。在产生的一系列点集和查询集上，我们进行了实验，将这两种策略与优化 kd 树策略进行了比较，并给对实验结果进行了分析。我们论证了：对于聚类数据和查询集的近似最邻近查询，这两种算法的效率较标准的 kd 树有显著提高。

1.介绍

最邻近查询需要解决的是这样一个问题：我们给定集合S，它含有度量空间

X中的n个点。我们要对这n个点进行处理，以便对给定的任意点q（q∈X）我们都能很快的找到S中距离它最近的点。最邻近查询在很多领域中都有应用，比如知识发现、数据挖掘、模式识别和分类、机器学习、数据压缩、多媒体数据库、文件恢复、统计等等。

对于度量空间，我们有许多种选择。我们始终假定空间为R^d，即d维空间，空间中的距离采用闵可夫斯基Lm距离进行度量。对于任意整数$m \geq 1$，R^d中任意两点$p = (p1, p2 \cdots, pd)$ 、$q = (q1, q2, \cdots, qd)$的距离被定义为$\sum_{1 \leq i \leq d} | p_i - q_i |^m$的m次方根。$L_1$，$L_2$和$L_\infty$分别被称为：曼哈顿距离，欧氏距离和最大距离。

我们将主要的焦点集中在数据结构上。因为数据集可能很大，我们将其做一下限制，只考虑数据空间程线性增长的情况。在众多方法之中，最流行的是基于分等级分解空间的方法。在这个领域中，最根本的工作是由Friedman,Bentley和Finkel完成的。他们指出通过使用kd树，可以将固定维数空间中可预计其分布的数据的空间复杂度控制在O(n)，查询的时间复杂度控制在O(logn)。对于这一问题，可选择的方法有许多种。但是，所有已知的方法都存在这样一个缺陷：当空间维数增加的时候，运行时间或空间复杂度将随之以指数倍增长。

我们试图找到一种高效的算法，即使在最糟的情况下，它也会有较低的空间复杂度和较短的查询时间，但事实证明，想要获得这样的算法是很困难的，但这同时也告诉我们寻找近似的最近临是解决问题的一条出路。设S为R^d中的点集，查询点q为R^d中任意一点，假定$\varepsilon > 0$，如果dist（p,q）$\leq$（1+ε）dist（p^*，q）成立，我们就说点p是点q的近似最近临。换句话说，p是在相对误差ε内的真实最近临。最近，人们已经对近似最近临问题进行了深入的研究。较有代表性的算法包括：Bern[Ber93],Arya和Mount[AM93b],Arya等人[AMN+98],Kleinberg[Kle97],Clarkson[Cla94], Chan[Cha97], Indyk和Motwani[IM98]以及Kushilevitz,Ostrovsky和Rabani[KOR98]。

本文中，我们以分级空间分解特别是kd树为基础，将数据结构的大小限定在O(dn)。这是因为在种数据结构较为简单而且应用广泛。kd树是一种二元树，它依靠垂直于坐标轴的分割超平面将多维空间划分为多级子空间。我们将在下一节对这个问题进行深入的探讨。设计kd树的一个关键问题是分割超平面的选择。Friedman,Bentley和Finkel提出了一种分割策略：假设点集在第k维

上有最大的延展度，则将过第k维中点且垂直于第k维坐标轴的平面作为分割超平面。他们将用这种方法得到的树称为优化kd树，我们把用这种方法叫作标准分割策略。另一种较为普遍的方法是利用包含点集的盒子形状（而非点集的分布）对点集划分。这种策略将垂直于盒子最长边且过最长边中点的平面作为分割超平面。我们将其称为中点分割策略。

以分级空间分解为基础，人们把出了许多用于最近临查询的数据结构。Yianilos引入了vp树。它不像kd树那样用一个平面去分割点集，而是利用一个称为优势点的数据点作为超球面的中心，将空间分为两部分。有一些基于R树及其变体的数据结构被应用于数据库领域。比如X树，它通过避免高度堆叠提高了R^*树的性能。类似的例子还有SR树。TV树在处理高维高间时使用了较为独特的方法。它通过维护一系列活跃维达到了降维的目的。当一个节点中的数据点共享了一个活动维上的相同的坐标时，该维将被列为无效，活跃维集合也将随之改变。

在这篇文章中我们研究了另外两种分割策略的的性能，并且将其与以往的kd树分割策略进行了比较。第一种策略叫做“滑动中点”，它是由Mount和Arya针对近似最近临搜索提出的，并应用于ANN函数库中。将这种方法引入函数库是为了更好的处理高聚性数据集。这种策略采用一种较为简单的方法来更正标准kd树分割策略中的一个严重缺陷。这个缺陷在于，当高聚性的数据点存在于低维子空间时，采用标准kd树分割策略对点集进行划分将会产生许多长宽比极高的盒子，这些盒子会使查询时间大大延长。“滑动中点”策略起初也是沿盒子的最长边进行简单的中点分割，但是，如果分割产生的任何子盒中不含数据点，该策略将使分割面沿点集沿伸方向滑动，直到遇到第一个数据点为止。在3.1节中我们将对这种分割策略进行详细描述并对其特性做出分析。

第二种策略叫做“最小不确定性”分割策略，它是一项基于询问的技术。建树时不仅要给出数据点集，还要选出一些示例性的查询点，这些点被称为训练点。建树时，算法应用了一种启发式的贪心算法，它试图使对训练集的查询时间最小。给定一个查询点集，我们将最邻近查询算法的每一步看作一张双向连通的图，称之为候选图，它的顶点代表了所有的数据点和查询点，图中与查询点紧挨着的数据子集可能就是该查询点的最邻近点。“最小不确定”策略在执行的每一步选出一个分割平面，尽可能多地删除候选图中剩余的边。在3.2节中，我们对这种策略进行了更为详细的描述。

我们在执行这两种策略的同时也执行了标准kd树分割策略。在产生的一系列有着各种分布的低维聚类数据上，我们对这些算法进行了比较。我们相信这

种类型的聚类数据在实际应用中的较为普遍的。我们使用手工合成的数据作为与基准数据的一个对比，以便于对我们聚类数据的密度和维数进行调整。我们的结果表明，对于低维聚类数据集，这两种算法的效率较标准的kd树有显著提高。以下的篇章是这样组织的，在下一章中我们展示了kd树的背景信息以及它是怎样执行最邻近查询的。在第3节中，我们对两个新的分割策略进行了描述。第四章，主要是对实验及结果的展示。

2.背景

在这节里我们描述了kd树是如何执行精确或近似的最邻近查询的。Bentley将kd树作为一种在高维空间中通用的二元搜索树[Ben75]。Kd树的每一个的节点可以看作是一个多维矩形，我们将基称为盒子。根节点是一个包含了所有数据点的矩形。其余的节点代表了包含数据点子集的矩形。（在两个矩形边界上的点可以认为是包含在其中任何一个矩形中）。如果节点中的数据点个数小于一个被称为bucket size的阈值，这个点就被称为叶节点，这些数据点就存储在叶节点中。否则，建立算法就选择一个分割超平面，它垂直于其中一个坐标轴，并且贯穿整个盒子。有许多分割策略可以用于超平面的选择。我们将在下面对此进行更详细的讨论。超平面将盒子划分为两个子矩形，相当于是当前节点的两个孩子节点，盒子中的数据点属于哪个孩子节点，取决于数据点在超平面的哪一侧。树中的每个内部节点都与它的分割超平面相关。

Fridman，Bentley和Finkel[FBF77]给出了一种用kd树寻找最邻近数据点的算法。他们所介绍了下面的分割策略，我们将面称为“标准分割策略”。对每个内部节点，我们将沿着点集的最大延展度方向选择分割平面，并使其垂直于某个坐标轴。分割点被选在中点，这样分割后形成的两个子集中含有的数据点数基本相同。最终得到的树的大小为O(n)，高度为O(logn)。White和Jain[WJ96]提出了另一种方法，称为VAM分割，两种方法采用的基本思想是一致的，但后者是将具有最大差异度的维定为分割维。

我们用一种简单的递归算法对查询进行回答。在最基本的情况下，当算法执行到叶节点时，它会对叶中的每个数据点与查询点间的距离进行计算。最小的距离值将被保存。当我们到达内部节点时，算法首先决定查询点位于超平面的哪一侧。查询点必须离即将被访问的孩子较近。算法递归地访问这个孩子。当返回时，算法判断盒子中另一个孩子与查询点的距离是否比当前最小距离要近，如果是，些也对这个孩子进行递归访问。当搜索返回到根时，我们就能得到最邻近点了。一个重要的观察结论是：对于任何查询点，算法将访问每一个

与查询点距离小与最邻近点的叶节点。

要对这种近似最邻近查询进行概括是较为简单的。设 ε 为允许的误差边界。在处理内部节点时，当且仅当先前距离较远的孩子节点与查询点的距离是当前最小距离的1/(1+ ε)时，我们才对这个孩子节点进行访问。Aray等人证明了这个过程的正确性。他们也展示了这种通用的算法是如何计算出k个精确或者近似邻近点的。

Arya和Mount[AM93a]提出了一些对这种算法的改进。第一种叫作“增量距离计算”。这是一种可以用于任何Minkowski距离计算的方法。除了分割超平面，树的每一个内部节点还存储了相关的盒子的信息。算法并不计算真正的距离，而是使用距离的平方。当算法执行到内部节点时，它计算查询点与相关盒子的平方距。他们指出在固定的时间内（不依赖维数），用这个信息计算出与每了孩子节点对应的盒子的平方距是完全可能的。他们还展示了一种叫作“优先搜索”的方法，这种方法使用堆，按照距离增加的顺序访问每个叶节点，而非按照树结构递归地访问。另外一种众所周知的用于最近邻查询的算法叫作部分距离计算[BG85,Spr91]。当计算查询点与数据点距离时，如果各平方量的和超过了当前最近点与查询点距离的平方，那么停止对该距离的计算。

对近似最邻近查询，Arya等人发现，对于任何一种基于分解空间的近似最邻近查询，都有两条重要的性质。

(1)平衡：树的高度应该为O(logn),其中n是数据点的个数。

(2)长宽比界限：树的叶节点的长宽比应该有一个限界，这意味着每个叶节点的最长边与最短边的比值应该有一个固定的上限值。

在这两条规则的基础上，他们又指出，最临近查询可以将数据节构的大小控制在 O（dn），将执行时间控制在 0（logn）。不幸的是，对于 kd 树而言，这两条性质不总是同时成立，特别是当数据点的分布特别集中时。Arya 等人提出了一种更复杂一些的数据结构，叫做“balanced box-decomposition”树，它可以同时满足这两条性质。这了证时他们的理论，这种额外的复杂性看来是有必要的，并且他们的实验也证明了当数据集是低维子空间上的高聚性数据时这种复杂性是很重要的。一个有趣但又实际的问题是，是否存在一种算法，它有着 kd 树的简单，又能对实际中的高聚分布的数据提供高效的搜索。

固定长宽比的上限，对高效搜索来说是充分非必要条件。一种更准确的应于他们的实验结果的条件被称为“packing constraint”。定义一个半径为r的球为点集的中心。“packing constraint”表明与球相交的盒子的个数是有限的。

Packing Constraint:大小至少为s的与半径为r的球相交的叶节点的个数是有上限的，它受限于的r/s，但不依赖于n。

如果树的盒子有长宽比限制，那么它一定满足“packing constraint”。Arya等人指出，如果满足“packing constraint”，那么这个盒子数只取决于维数和ε。标准分割策略的不足在于它不能控制盒子的长宽比，因此不能完全满足“packing constraint”。

附 录 2

Analysis of Approximate Nearest Neighbor Searching with Clustered Point Sets

Songrit Maneewongvatana and David M. Mount

Abstract

Nearest neighbor searching is a fundamental computational problem. A set of *n* data points is given in real *d*-dimensional space, and the problem is to preprocess these points into a data structure, so that given aquery point, the nearest data point to the query point can be reported efficiently. Because data sets can be quite large, we are primarily interested indata structures that use only $O(dn)$ storage.

A popular class of data structures for nearest neighbor searching is the kd-tree and variants based on hierarchically decomposing space into rectangular cells. An important question in the construction of such data structures isthe choice of a *splitting method*, which determines the dimension and splitting plane to be used at each stage of the decomposition. This choice of splitting method can have a signi_cant influence on the e_ciency of the data structure.This is especially true when data and query points are clustered in low dimensional subspaces. This is because clustering can lead to subdivisions in which cells have very high aspect ratios.

We compare the well-known optimized kd-tree splitting method against two alternative splitting methods. The first, called the *sliding-midpoint* method,which attempts to balance the goals of producing subdivision cells of bounded aspect ratio, while not producing any empty cells. The second, called the *minimum-ambiguity* method is a query-based approach. In addition to the data points, it is also given a training set of query points for preprocessing.It employs a simple greedy algorithm to select the splitting plane that minimizes the average amount of ambiguity in the choice of the nearest neighbor for the training points. We provide an empirical analysis comparing these two methods against the optimized kd-tree construction for a number of synthetically generated data and query sets. We demonstrate that for clustered data and query sets, these algorithms can provide signi_cant improvements over the standard kd-tree construction for approximate nearest neighbor searching.

1. Introduction

Nearest neighbor searching is the following problem: we are given a set *S* of *n data points* in a metric space, *X*, and are asked to preprocess these points so that, given any *query point* $q \in X$, the data point nearest to *q* can be reported quickly. Nearest neighbor searching has applications in many areas, including knowledge discovery and data mining [**FPSSU96**], pattern recognition and classification [**CH67, DH73**], machine learning [**CS93**], data compression [**GG92**], multimedia databases [**FSN+95**], document retrieval [**DDF+90**], and statistics [**DW82**].

There are many possible choices of the metric space. Throughout we will assume that the space is R^d, real *d*-dimensional space, where distances are measured using any Minkowski *Lm* distance metric. For any integer $m \geq 1$, the *Lm-distance* between points $p = (p1, p2 \cdots, pd)$ and $q = (q1, q2, \cdots, qd)$ in R^d is defined to be the *m*-th root of $\sum_{1 \leq i \leq d} | p_i - q_i |^m$. The *L*1, *L*2, and L_∞ metrics are the well-known Manhattan, Euclidean and max metrics, respectively.

Our primary focus is on data structures that are stored in main memory. Since data sets can be large, we limit ourselves to consideration of data structures whose total space grows linearly with *d* and *n*. Among the most popular methods are those based on hierarchical decompositions of space. The seminal work in this area was by Friedman, Bentley, and Finkel [**FBF77**] who showed that *O*(*n*) space and *O*(log *n*) query time are achievable for fixed dimensional spaces in the expected case for data distributions of bounded density through the use of kd-trees. There have been numerous variations on this theme. However, all known methods suffer from the fact that as dimension increases, either running time or space increase exponentially with dimension.

The difficulty of obtaining algorithms that are e_cient in the worst case with respect to both space and query time suggests the alternative problem of finding *approximate* nearest neighbors. Consider a set *S* of data points in *Rd* and a query point $q \in R^d$. Given $\varepsilon > 0$, we say that a point $p \in S$ is a $(1 + \varepsilon)$-*approximate nearest neighbor* of *q* if

$$dist\ (p,q) \leq (1+\varepsilon)\ dist\ (p^*, q)$$

where p^* is the true nearest neighbor to *q*. In other words, *p* is within relative error ε of the true nearest neighbor. The approximate nearest neighbor problem has been heavily studied recently. Examples include algorithms by Bern [**Ber93**], Arya and Mount [**AM93b**], Arya, et al. [**AMN+98**], Clarkson [**Cla94**], Chan [**Cha97**],Kleinberg [**Kle97**], Indyk and Motwani [**IM98**], and Kushilevitz, Ostrovsky and Rabani [**KOR98**].

In this study we restrict attention to data structures of size $O(dn)$ based on hierarchical spatial decompositions, and the kd-tree in particular. In large part this is because of the simplicity and widespread popularity of this data structure.
A kd-tree is binary tree based on a hierarchical subdivision of space by splitting hyperplanes that are orthogonal to the coordinate axes [**FBF77**]. It is described further in the next section. A key issue in the design of the kd-tree is the choice of the splitting hyperplane. Friedman, Bentley, and Finkel proposed a splitting method based on selecting the plane orthogonal to the median coordinate along which the points have the greatest spread. They called the resulting tree an optimized kd-tree, and henceforth we call the resulting splitting method the *standard splitting method*. Another common alternative uses the shape of the cell, rather than the distribution of the data points. It splits each cell through its midpoint by a hyperplane orthogonal to its longest side. We call this the *midpoint splitmethod*.

A number of other data structures for nearest neighbor searching based on hierarchical spatial decompositions have been proposed. Yianilos introduced the *vp-tree* [**Yia93**]. Rather than using an axis-aligned plane to split a node as in kd-tree, it uses a data point, called the vantage point, as the center of a hypersphere that partitions the space into two regions. There has also been quite a bit of interest from the _eld of databases. There are several data structures for database applications based on *R*-trees and their variants [**BKSS90, SRF87**]. For example, the *X-tree* [**BKK96**] improves the performance of the R^*-tree by avoiding high overlap. Another example is the SR-tree [**KS97**]. The *TV-tree* [**LJF94**] uses a different metherd to deal with high dimensional spaces. It reduces dimensionality by maintaining a number of active dimensions. When all data points in a node share the same coordinate of an active dimension, that dimension will be deactivated and the set of active dimensions shifts.

In this paper we study the performance of two other splitting methods, and

compare them against the kd-tree splitting method. The first, called *sliding-midpoint*, is a splitting method that was introduced by Mount and Arya in the ANN library for approximate nearest neighbor searching [**MA97**]. This method was introduced into the library in order to better handle highly clustered data sets. We know of no analysis (empirical or theoretical) of this method. This method was designed as a simple technique for addressing one of the most serious flaws in the standard kd-tree splitting method. The flaw is that when the data points are highly clustered in low dimensional subspaces, then the standard kd-tree splitting method may produce highly elongated cells, and these can lead to slow query times. This splitting method starts with a simple midpoint split of the longest side of the cell, but if this split results in either subcell containing no data points, it translates(or "slides") the splitting plane in the direction of the points until hitting the firrst data point. In Section 3.1 we describe this splitting method and analyze some of its properties.

The second splitting method, called *minimum-ambiguity*, is a query-based technique. The tree is given not only the data points, but also a collection of sample query points, called the *training points*. The algorithm applies a greedy heuristic to build the tree in an attempt to minimize the expected query time on the training points. We model query processing as the problem of eliminating data points from consideration as the possible candidates for the nearest neighbor. Given a collection of query points, we can model any stage of the nearest neighbour algorithm as a bipartite graph, called the *candidate graph*, whose vertices correspond to the union of the data points and the query points, and in which each query point is adjacent to the subset of data points that might be its nearest neighbor. The minimum-ambiguity selects the splitting plane at each stage that eliminates the maximum number of remaining edges in the candidate graph. In Section 3.2 we describe this splitting method in greater detail.

We implemented these two splitting methods, along with the standard kd-tree splitting method. We compared them on a number of synthetically generated point distributions, which were designed to model low-dimensional clustering. We believe this type of clustering is not uncommon in many application data sets [JD88]. We used synthetic data sets, as opposed to standard benchmarks, so that we could adjust the strength and dimensionality of the clustering. Our results sh-

ow that these new splitting methods can provide signi_cant improvements over the standard kd-tree splitting method for data sets with low-dimensional cluster-ing .The rest of the paper is organized as follows. In the next section we present background information on the kd-tree and how to perform nearest neighbor sea- rches in this tree. In Section 3 we present the two new splitting methods. In Section 4 wedescribe our implementation and present our empirical results.

2. Background

In this section we describe how kd-trees are used for performing exact and approximate nearest neighbor searching. Bentley introduced the kd-tree as a gen-eralization of the binary search tree in higher dimensions [**Ben75**]. Each node of the tree is implicitly associated with a *d*-dimensional rectangle, called its *cell*. The root node is associated with the bounding rectangle, which encloses all of the data points. Each node is also implicitly associated with the subset of data points that lie within this rectangle. (Data points lying on the boundary between two rectangles, may be associated with either.) If the number of points associated with a node falls below a given threshold, called the *bucket size*, then this node is a leaf, and these points are stored with the leaf. (In our experiments we used a bucket size of one.) Otherwise, the construction algorithm selects a splitting hyp-erplane, which is orthogonal to one of the coordinate axes and passes through the cell. There are a number of *splitting methods* that may be used for choosing this hyperplane. We will discuss these in greater detail below. The hyperplane subdi-vides the associated cell into two subrectangles, which are then associated with the children of this node,and the points are subdivided among these children according to which side of the hyperplane they lie. Each internal node of the tree is associated with its splitting hyperplane (which may be given as the index of the orthogonal axis and a cutting value along this axis).

Friedman, Bentley and Finkel [**FBF77**] present an algorithm to _nd the near-est neighbor using the kd-trees. They introduce the following splitting method, which we call the *standard splitting method*. For each internal node, the splitting hyperplane is chosen to be orthogonal to the axis along which the points have the greatest *spread* (di_erence of maximum and minimum). The splitting point is ch-osen at the median coordinate, so that the two subsets of data points have nearly-qual

sizes. The resulting tree has $O(n)$ size and $O(\log n)$ height. White and Jain [**WJ96**] proposed an alternative, called the *VAM-split*, with the same basic idea, but the splitting dimension is chosen to be the one with the maximum variance.

Queries are answered by a simple recursive algorithm. In the basis case, when the algorithm arrives at a leaf of the tree, it computes the distance from the query point to each of the data points associated with this node. The smallest such dist-tance is saved. When arriving at an internal node, it _rst determines the side of the associated hyperplane on which the query point lies. The query point is nece-ssarily closer to this child's cell. The search recursively visits this child. On retu-rning from the search, it determines whether the cell associated with the other child is closer to the query point than the closest point seen so far. If so, then this child is also visited recursively. When the search returns from the root, the clos-est point seen is returned. An important observation is that for each query point, every leaf whose distance from the query point is less than the nearest neighbor will be visited by the algorithm.

It is an easy matter to generalize this search algorithm for answering *approx-imate* nearest neighbor queries. Let ε denote the allowed error bound. In the processing of an internal node, the further child is visited only if its distance from the query point is less than the distance to the closest point so far, divided by $(1 + \varepsilon)$. Arya et al. [**AMN+98**] show the correctness of this procedure. They also show how to generalize the search algorithm for computing the k-closest neighbors,either exactly or approximately.

Arya and Mount [**AM93a**] proposed a number of improvements to this basic algorithm. The first is called *incremental distance calculation*. This technique can be applied for any Minkowski metric. In addition to storing the splitting hyperplane, each internal node of the tree also stores the extents of associated cell projected orthogonally onto its splitting axis. The algorithm does not main-tain true distances, but instead (for the Euclidean metric) maintains squared distances.When the algorithm arrives at an internal node, it maintains the squared distancefrom the query point to the associated cell. They show that in constant time(independent of dimension) it is possible to use this information to compute thesquared distance to each of the children's cell. They also presented a method called *priority search*, which uses a heap to visit the leaves of the tree in increas-ing

order of distance from the query point, rather than in the recursive order dica-tated by the structure of the tree. Yet another improvement is a well-known tech-nique from nearest neighbor searching, called *partial distance calculation* [**BG85, Spr91**].When computing the distance between the query point and a data point, if the accumulated sum of squared components ever exceeds the squared distance to the nearest point so far, then the distance computation is terminated.

One of the important elements of approximate nearest neighbor searching, which was observed by Arya et al. [**AMN+98**], is that there are two important properties of any data structure for approximate nearest neighbor searching bas-ed on spatial decomposition.

Balance:: The height of the tree should be $O(\log n)$, where n is the number of data points.

Bounded aspect ratio:: The leaf cells of the tree should have bounded aspect-ratio, meaning that the ratio of the longest to shortest side of each leaf cell should be bounded above by a constant.

Given these two constraints, they show that approximate nearest neighbor searching (using priority search) can be performed in $O(\log n)$ time from a data structure of size $O(dn)$. This is particularly true when the point distribution is highly clustered. Arya et al. present a somewhat more complex data structure called a *balanced box-decomposition tree*, which does satisfy these properties. The extra complexity seems to be necessary in order to prove their theoretical results, and they show empirically that it is important when data sets are highly clustered in low-dimensional subspaces. An interesting practical question is whether there exist methods that retain the essential simplicity of the kd-tree, while providing practical efficiency for clustered data distributions (at least in most instances, if not in the worst case).

Bounded aspect ratio is a su_cient condition for e_ciency, but it is not nec-essary. The more precise condition in order for their results to apply is called the *packing constraint* [**AMN+98**]. Define a *ball* of radius r to be the locus of points that are within distance r of some point in Rd according to the chosen metric. The packing constraint says that the number of large cells that intersect any such ball is bounded.

Packing Constraint:: The number of leaf cells of size at least s that intersect an open ball of radius $r > 0$ is bounded above by a function of r/s and d, but inde-

pendent of *n*.

If a tree has cells of bounded aspect ratio, then it satisfies the packing constraint. Arya et al., show that priority search runs in time that is proportional to the depth of the tree times the number of cells of maximum side length $r\varepsilon/d$ that intersect a ball of radius *r*. If the packing constraint holds, then this number of cells depends only on the dimension and ε . The main shortcoming of the stand-ard splitting method is that it may result in cells of unbounded aspect ratio, and hence does not generally satisfy the packing constraint.