第 44 卷 第 6 期 计 算 机 工 程 2018 年 6 月
Vol. 44 No. 6 Computer Engineering June 2018

·安全技术· 文章编号：1000-3428(2018)06-0117-05 文献标志码：A 中图分类号：TP309

源代码中的 API 密钥自动识别方法

薛 敏[1]，方 勇[2]，黄 诚[1]，刘 亮[2]

（1. 四川大学 电子信息学院，成都 610065；2. 四川大学 网络空间安全学院，成都 610207）

摘 要： 应用程序编程接口（API）密钥的泄露可能导致相关服务被恶意利用，从而造成难以预估的经济损失。为此，通过对样本进行基本特征统计和源代码静态结构分析，提取出不同项目代码中 API 密钥的共性特征，从而构建一种基于机器学习的自动识别源代码中 API 密钥的方法。实验结果表明，该识别方法的检索性能比全文匹配搜索、关键字搜索和信息熵值搜索等传统检测方式更优。

关键词： 应用程序编程接口密钥；源代码；机器学习；静态结构；信息熵

中文引用格式： 薛 敏，方 勇，黄 诚，等. 源代码中的 API 密钥自动识别方法[J]. 计算机工程，2018，44(6)：117-121，129.

英文引用格式： XUE Min，FANG Yong，HUANG Cheng，et al. Automatic identification method of API key in source code[J]. Computer Engineering，2018，44(6)：117-121，129.

Automatic Identification Method of API Key in Source Code

XUE Min[1]，FANG Yong[2]，HUANG Cheng[1]，LIU Liang[2]

（1. College of Electronics and Information，Sichuan University，Chengdu 610065，China；
2. College of Cybersecurity，Sichuan University，Chengdu 610207，China）

【Abstract】 The leak of Application Programming Interface（API） key may cause the illegal use of services，and then lead to unpredictable economic losses. The common characteristics of API keys in different project codes are extracted by analyzing the basic characteristics statistics and the source code static structure of the samples. Then，an automatic identification method based on machine learning is built to detect the API keys in the source code. The result of 10-fold cross-validation experiment results show that the identification method is better in retrieval performance than traditional detection approaches such as full-text matching search，keywords search and information entropy search.

【Key words】 Application Programming Interface（API） key；source code；machine learning；static structure；information entropy

DOI： 10.19678/j.issn.1000-3428.0048197

0 概述

应用程序编程接口（Application Programming Interface，API）密钥通常被用作唯一标识符和秘密令牌，用以认证 API 调用者的身份，以及跟踪和控制 API 的调用方式。大量网页应用和移动端应用会使用一些外部服务，比如 Google Map、Facebook、Twitter 等，它们之间的认证机制便是通过 API 密钥[1]。大多数情况下，API 密钥是以硬编码的形式存在于客户端应用程序[2]中，一旦被盗取，恶意用户就可以以开发者的身份免费使用相关服务，进而造成相关服务被滥用。

2013 年，上万个 Amazon Web Services（AWS）密钥被发现上传至开源代码仓库 GitHub。而一些 API 密钥已被恶意盗用，给密钥的拥有者造成了大量经济损失[3]。2016 年 11 月，Fallible 网络安全公司检测了 16 000 多个 Android 应用程序后发现，其中 304 个应用程序包含允许访问机密或关键系统的 API 密钥[4]。例如，被发现的 Uber（一种打车应用）密钥可用于发送 Uber 系统通知。

Dylan Ayrey 开发的 Truffle Hog[5] 工具通过检索 GitHub 代码库的代码提交记录和分支，结合密钥的高熵特性，从而检测出疑似 API 密钥，但由于判定机制单一，导致其误报率较高。文献[6]对比了简单模式匹配、启发式过滤和基于源码程序切片等方式的检测效果，发现上述几种方式结合的检测方法能达到 100% 的正确率，但这只是针对 AWS 密钥和 Facebook 密钥的检测效果，而且样本数分别为 84 和 30，相对偏少。

本文基于以上研究基础，设计并实现基于机器

作者简介：薛 敏（1993—），女，硕士研究生，主研方向为 Web 安全；方 勇，教授、博士；黄 诚，博士；刘 亮，讲师、博士。

收稿日期：2017-07-31 修回日期：2017-09-01 E-mail：simmin_x@163.com

学习的源码中 API 密钥的自动识别模型,用以帮助开发人员在项目代码部署或开源之前及时发现隐藏的 API 密钥,避免造成不必要的损失。同时提出针对不同编程语言的项目代码中 API 密钥共性特征的构建方法。该共性特征包括 API 密钥基本特征、区别于普通源码字符串的个性特征、表征其单字符之间关联度的随机性特征,以及通过属性特征向量计算其代码相似度的源码结构特征。在已有研究基础上,提出基于随机森林的源代码中 API 密钥自动识别模型。通过综合分析 API 密钥的生成规则、表现形式、随机性特征以及其在源码中的表现,从而实现 API 密钥的检测。

1 源代码中的 API 密钥特征提取

本文首先通过关键字搜索从 GitHub 上爬取项目样本代码,然后进行人工筛选和分类标记。基于这些样本,做如下分析。

1.1 API 密钥的生成规则分析

API 密钥具有唯一标识授权 API 用户和无法被猜解、伪造等属性,因此,创建 API 密钥的一般方法是对用户应用程序的详细信息或者随机字符串进行哈希计算。还有一些会使用通用唯一识别码(Universally Unique Identifier,UUID)或者全局唯一标识符(Globally Unique Identifier,GUID)。更多地,一些开发人员会选择在此基础上再进行一次 Base-64 编码。

由于不同应用程序提供的 API 名称千差万别,不同编程语言对 API 的调用也不尽相同,因此本文从 API 密钥生成规则以及表现形式出发,提出了基于不同项目中 API 密钥在表现上的共性特征与独立于其他代码字符串的特征计算方法。代码如下所示:

```
public static BasicAWSCredentials authenticateToEC2()
{
String ACCESS _ KEY _ ID = " AKIAJSLPDBUKPU66NUKA";
String SECRET_KEY =
"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";
BasicAWSCredentials awsCreds =
new BasicAWSCredentials(ACCESS_KEY_ID,SECRET_KEY);
  return awsCreds;
}
```

该程序段为 Java 程序中使用 AWS 密钥的示例,变量 SECRET_KEY 的值即为 API 密钥。

1.2 API 密钥特征提取

本文通过对提取到的样本进行研究分析,提取基本特征、随机性特征和源码结构特征 3 种特征。

1.2.1 基本特征

从 API 密钥生成的规则和表现上看,其字符串除了比一般源码字符串长之外,还偶尔包含 1 个 ~ 2 个特殊字符,数字也在字母中穿插出现,但数量占比不高。更重要的是,API 密钥字符串是难以拼读的,正常的语句是元辅音交替出现,对于相同长度的 API 密钥字符串 s_1 和正常语句 s_2,s_1 中元音字符占比会低于 s_2。为了验证这个猜想,先对样本 API 密钥进行字符串长度分布统计,发现样本中占比前 5 的字符串长度依次为 32、44、40、50、27,如表 1 所示。

表 1 样本中的 API 密钥长度分布

字符串长度	数量占比/%
32	25.688
44	18.654
40	11.621
50	6.422
27	6.116

再将正常文本分别按这 5 个长度进行去空格、去标点符号后拆分,以元音字符占比的平均值与相应长度 API 密钥的元音字符占比做比较,结果如图 1 所示。

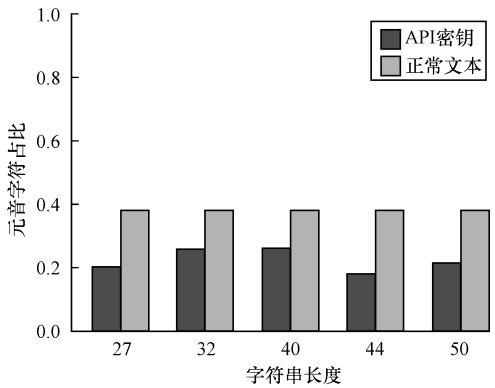

图 1 API 密钥与正常文本中元音字符占比比较

由图 1 可以看出,对于正常文本,在取较长字符串时,其元音字符占比稳定在 0.382 左右,而对于 API 密钥,其元音字符占比虽然未恒定在某一个值,但均比正常文本偏小。

基于以上分析,将如表 2 所示的 4 种特征作为 API 密钥的基本特征。

表 2 API 密钥基本特征构成

特征	描述
字符串长度	字符串中所有字符的数量
特殊字符占比	字符串中除字符和数字外的字符都属于特殊字符,例如“+”“/”等
数字占比	字符串中数字字符数量所占比例
元音字符占比	字符串中元音字符数量所占比例(去除数字和特殊字符)

1.2.2　随机性特征

API 密钥之所以难以拼读,不只因为其元音字符占比偏小,更多的是因为相邻字母之间的关联程度低。本文采用信息熵和对数似然 2 种计算方式对字母之间这种随机性特征进行提取。

1)信息熵

1984 年,香农提出“信息熵”[7]的概念,将其用于衡量信息量大小,信息的不确定性越高,信息量越大,信息熵也就越高。API 密钥字符串随机性高,难以拼读,其信息熵值偏高[8]。信息熵值计算公式为:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\,\mathrm{lb}\,p(x_i) \tag{1}$$

本文利用上述公式计算一个字符串所携带的信息量,其中,X 表示该字符串,x_i 表示 X 中第 i 个字符,$p(x_i)$ 表示 x_i 在 X 中出现的频率。

部分计算代码如下:

```
/*定义迭代器构成*/
1. iterator = " ABCDEFGHIJKLMNOPQRSTU
    VWXYZabcdefghijklmnopqrstuvwxyz
    0123456789 +/= -"
/*循环读取迭代器中的每个字符,计算迭代器中每个字符在字符串中出现的频率*/
2.   en_val = 0
3.   for item in iterator:
4.       pItem = float( str. count( item ) )/len( str)
5.       if pItem > 0:
6.           en_val + = -pItem * math. log( -pItem,2);
7.   return en_val
```

将部分样本项目代码预处理后,将提取的字符串信息熵与 API 密钥样本信息熵作对比,如图 2 所示。

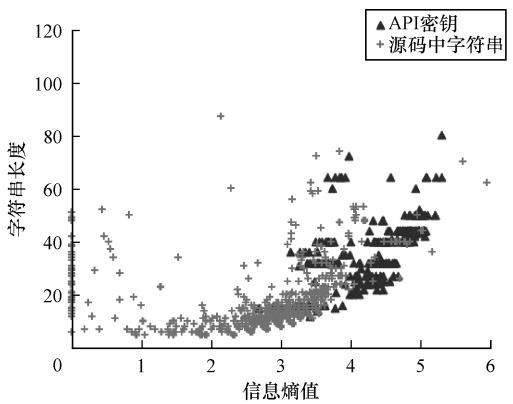

图 2　API 密钥与普通源码字符串信息熵值比较

由图 2 可以看出,虽然项目代码中的字符串信息熵分布有高有低,但是大部分比 API 密钥信息熵低,而且 API 密钥的熵值比较集中。

2)对数似然

本文使用一阶马尔可夫模型[9]作为英文字符的概率模型,计算每个字符串中英文字符序列的对数似然。为了定义字符之间的转换概率,从 Google Ngrams 语料库[10]中的部分单位列表中计算出一阶转换矩阵。对于每个字符串,首先删除数字和特殊字符,然后使用字符序列中第一个字符在单位列表中的概率分布作为初始概率向量,计算剩余字符序列的对数似然[11]。最后将对数似然值除以字符串长度(省略数字和连字符),以说明平均较长的字符串具有较低的对数似然性。

在一阶马尔可夫模型中,假设一个当前字符只与它前一个字符有关,例如,字符串“character”中“h”出现的概率只与它前面的“c”有关。对于省略数字和特殊字符后的长度为 n 的字符串 $X=(x_1,x_2,\cdots,x_n)$,字符 x_n 的概率为 $p(x_n|x_{n-1})$,那么:

$$P(X)=p(x_1)p(x_2|x_1)\cdots p(x_n|x_{n-1})=\prod_{i=1}^{n}p(x_i|x_{i-1}) \tag{2}$$

由于 API 密钥字符串中字符之间后继概率很低,最后计算出的 $P(X)$ 极小,以至于计算机内部没有足够的位来表示,而将其大致归零。为了改善这种数值下溢问题,本文进一步采用对数似然估计进行处理。似然估计在数值上与相应概率相等,而对数处理也不影响原似然估计的单调性。因此,最后的对数似然估计计算方式为:

$$L(\theta|x)=\frac{\ln P(X)}{n}=\frac{\ln\prod_{i=1}^{n}P(x_i|x_{i-1})}{n} \tag{3}$$

1.2.3　源码结构特征

本文通过分析 API 密钥所在的源码文件发现,API 密钥在程序结构和源码表现上有以下 6 个特点:

1)多以变量名-标量值、键值对等赋值形式出现。

2)嵌套层次较低。若给每个代码行赋予一个深度值,那么由于 API 密钥的赋值一般无需用到 if、while 等稍复杂的结构,因此其嵌套层次对于整体代码结构来说相对较低。

3)包含特殊变量名。API 密钥的变量名一般会包含 access、secret、key、token 等单词。

4)影响行数少。API 密钥一旦被赋值,几乎不会被更改,部分会被稍作变换,影响行数一般不超过 3 行。

5)调用次数为 0 或者 1。0 次的情况是指 API 密钥直接作为实参被调用或者在当前源码中未被调用;1 次的情况是指 API 密钥先被定义,再被调用。

6)所在行数靠前。从一般编程习惯上看,开头部分是导入其他包程序,然后是正文部分,而对常量的赋值会在正文的前面部分,因此 API 密钥所在行数相对靠前。

基于以上分析,本文定义了如表 3 所示的六维特征属性,对 726 个包含 API 密钥的文件进行属性计数[12],根据计数结果设置经验阈值。

表 3 源码结构属性特征

特征属性	离散值
是否赋值、键值结构	是、否
嵌套层次	≤3、>3
包含特殊变量名	是、否
字符串出现次数	≤2、>2
变量名出现次数	≤2、>2
所在行数	≤20、>20

单个属性特征不足以代表 API 密钥所在源码的结构特性,因此,本文以上述 6 个属性特征离散值构建向量模型,利用余弦公式来度量待检测样本与目标样本之间的相似度[13]。目标对比向量 $\boldsymbol{V}=(v_1,v_2,\cdots,v_n)$(本文中 $\boldsymbol{V}=(1,1,\cdots,1)$),待检测代码向量 $\boldsymbol{W}=(w_1,w_2,\cdots,w_n)$,相似度 $Sim(\boldsymbol{V},\boldsymbol{W})$ 计算方式如下:

$$Sim(\boldsymbol{V},\boldsymbol{W})=\frac{\sum_{i=1}^{n}v_i\times w_i}{\sqrt{\sum_{i=1}^{n}(v_i)^2}\times\sqrt{\sum_{i=1}^{n}(w_i)^2}} \tag{4}$$

2 源代码中 API 密钥自动识别模型

根据前文对样本数据特征的研究和分析,本文构建一种基于随机森林算法的 API 密钥自动识别模型。

2.1 随机森林基本原理

随机森林是一种基于统计学习理论的机器学习算法,在具体应用中能够较好地容忍异常和噪声[14]。

随机森林是由 k 棵决策树 $\{h(\boldsymbol{x}|\boldsymbol{\theta}_k)\}$ 组成的组合分类器,其中,$\boldsymbol{x}$ 表示输入的特征向量,$\boldsymbol{\theta}_k$ 表示 k 个独立分布的随机向量 $\boldsymbol{\theta}_1,\boldsymbol{\theta}_2,\cdots,\boldsymbol{\theta}_k$。首先以 Boostrap 抽样方式从原始输入的训练集中有放回地随机抽取 k 个训练集,这 k 个训练集的样本容量与原始训练集一样;然后从待选特征集中随机选取一定数目的特征,根据性能划分最优的情形进行特征集的选取;最后利用 Bagging 方式建立 k 棵子决策树,根据每棵子树的分类结果进行投票,投票数最多的为最终分类结果。上述随机森林过程可由如下公式表示:

$$H(\boldsymbol{x})=\arg\max_{Y}\sum_{i=1}^{k}I(h(\boldsymbol{x}|\boldsymbol{\theta}_i)=Y) \tag{5}$$

其中,$H(\boldsymbol{x})$ 表示组合分类器,$h(\boldsymbol{x}|\boldsymbol{\theta}_i)$ 表示单个决策树分类模型,Y 表示输出变量,$I(\cdot)$ 表示指示函数。

2.2 自动识别模型

基于第 1.2 节中对样本的分析及特征提取,本文提出的基于随机森林的源代码中 API 密钥自动识别模型如图 3 所示。

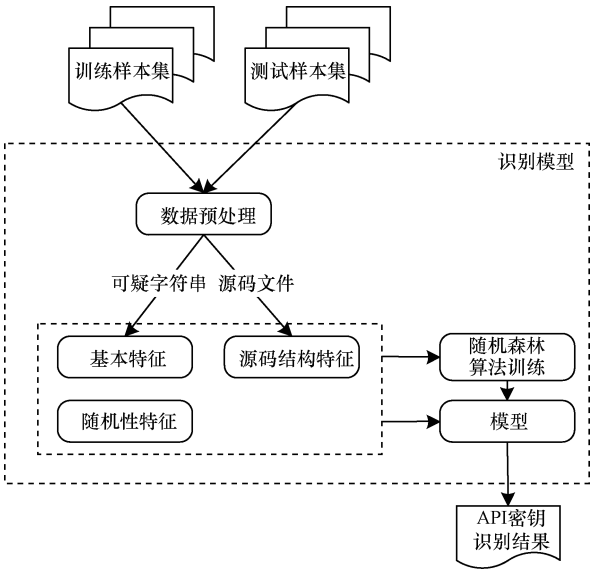

图 3 源代码中的 API 密钥自动识别模型结构

该模型由 3 个主体步骤组成:

1)数据预处理。本文从项目角度出发,检测可能包含在项目代码中的 API 密钥,因此,需要梳理出项目中的代码文件(基于后缀名,例如.java),再根据以下规则进一步筛选出可疑字符串:(1)正则匹配长度范围在[5,64]之间的不包含空格的字符串;(2)剔除纯数字、纯特殊字符的字符串;(3)剔除包含!、@、#、%、&、*等特殊字符的字符串。

2)特征提取。以 Google Ngrams 语料库中的部分单位列表为样本,训练生成一阶马尔科夫转换矩阵。预处理训练样本后,对提取到的字符串进行基本特征、随机性特征提取,然后对于字符串所在的源码文件,利用属性计数的方式,结合余弦距离计算代码属性相似度,从而提取源码结构特征。由于 3 类特征值的值域不一样,为了降低值域不同对分类结构造成的影响,再对特征值进行归一化处理。

3)分类算法。本文采用随机森林算法对选取的 3 类共 7 个特征进行训练,构造出多棵子决策树,以此构建组合分类器。对于每一棵子树的训练都采用随机有放回抽样的方式,防止出现过度拟合的情况[15]。设对 API 密钥的判定结果为 $Y\in\{0,1\}$,其中 0 表示非 API 密钥,1 表示 API 密钥,输入向量 $\boldsymbol{X}=\{x_1,x_2,\cdots,x_n\}$,共 n 个特征,本文中 $n=7$。分类算法的流程为:

(1)基于训练数据集进行 Boostrap 抽样[16],产生 k 个容量大小一致的训练子集 $S_1,S_2,\cdots,S_k$。

(2)根据第 2.1 节中的随机森林构造原理生成 k 个子决策树 $\{h(\boldsymbol{x}|\boldsymbol{\theta}_k)\}$。在此过程中,随机从 3 类共 7 个特征中选取 m 个特征,并依据基尼系数选取最优特征子集,直到各个子决策树完全生成。

(3)当输出新的待测样本时,各个子决策树对分类结果进行投票,最后以投票结果作为模型的分类结果。

3　实验结果及评估

本文通过十折交叉验证[17]方法对识别模型的准确性进行验证,并对比传统识别方法的检测结果,以证明模型的有效性。

3.1　实验数据

通过关键字搜索从 GitHub 上爬取项目样本代码,然后进行人工筛选和分类标记,最后得到包含 API 密钥的项目样本 688 个。其中,文件共计 8 326 个,包含 API 密钥的文件 726 个,API 密钥 847 个。

3.2　模型准确度测试

为了评价模型的准确度,采用十折交叉验证方法,并通过 P(精确率, Precision)、R(召回率, Recall)、F1 值[18] F_1(P 和 R 的调和平均数)来描述测试结果。

$$F_1 = \frac{2 \cdot P \cdot R}{P + R} \tag{6}$$

实验以 688 个包含 API 密钥的项目文件为正样本,以 2 473 个不包含 API 密钥的项目文件为负样本。为了更好地对比 3 类特征对识别效果的影响,将特征组合为 3 组进行实验,其中,B 表示基本特征,R 表示随机性特征,S 表示源码结构特征。每组实验将样本集分成 10 份,轮流选择其中 9 份作为训练数据,剩余的 1 份作为测试数据,进行验证。分类结果如表 4 所示,其中,$\overline{T}$ 表示真正率(预测为正的正样本比率)的平均值,$\overline{F}$ 表示假正率(预测为正的负样本比率)的平均值。

表 4　模型识别效果

特征	$\overline{T}$	$\overline{F}$	P	R	F_1
B	0.930	0.086	0.933	0.930	0.931
B + R	0.958	0.045	0.959	0.958	0.958
B + R + S	0.959	0.048	0.960	0.959	0.960

3.3　模型有效性验证

为了验证模型的有效性,对比了 3 种传统检测方法的检测效果:1)基于全文匹配搜索(F)方法,主要根据字符串长度、是否连续等特征构造正则表达式进行全文本检索;2)基于关键字搜索(K)方法,根据常用变量名 SECRET_KEY、Java 语言中 AWS 相关类名 BasicAWSCredentials 和方法 setClientSecret 等关键字进行检索;3)基于信息熵值搜索(E)方法。实验同样以 688 个包含 API 密钥的项目文件为正样本,以 2 473 个不包含 API 密钥的项目文件为负样本。对比实验的结果如表 5 所示。

表 5　本文方法与传统方法检测结果对比

检测方式	精确率	召回率
F	0.364	1.000
F + K	0.857	0.872
F + K + E	0.916	0.951
本文方法	0.960	0.959

从表 5 可以看出,以 3 种传统检测方式相结合进行检测,精确率可以达到 0.916,召回率达到 0.951。而使用本文提出的识别模型进行检测,精确率可以达到 0.960,召回率达到 0.959。实验结果证明,本文提出的源码中 API 密钥识别方法的检测结果优于传统检测方法。

4　结束语

本文从源代码静态结构和 API 密钥表现特点上分析样本特征,统计分析字符长度、数字占比、特殊字符占比、元音字符占比等基本特征,利用信息熵、对数似然估计等表征样本的随机性,从程序结构和源码表现方面测量源码结构的相似度,最后取得精确率为 0.960、召回率为 0.959 的检测效果。下阶段将尝试更细致全面的数据预处理,以及从字符串控制流、数据流 2 个方面进行特征分析。

参考文献

[1] FARRELL S. API keys to the kingdom [J]. IEEE Internet Computing, 2009, 13(5): 91-93.

[2] VIENNOT N, GARCIA E, NIEH J. A measurement study of google play [C]//Proceedings of ACM International Conference on Measurement & Modeling of Computer Systems. New York, USA: ACM Press, 2014: 221-233.

[3] Techspot. 10 000 AWS secret access keys carelessly left in code up-loaded to GitHub [EB/OL]. [2017-07-31]. http://www.techspot.com/news/56127-10000-aws-secret-access-keys-carelessly-left-in-code-uploaded-to-github.html.

[4] We reverse engineered 16k apps, here's what we found [EB/OL]. [2017-07-31]. https://hackernoon.com/we-reverse-engineered-16k-apps-heres-what-we-found-51bdf3b456bb.

[5] "Truffle Hog" tool detects secret key leaks on GitHub [EB/OL]. [2017-07-31]. http://www.securityweek.com/truffle-hog-tool-detects-secret-key-leaks-github.

[6] SINHA V S, SAHA D, DHOOLIA P, et al. Detecting and mitigating secret-key leaks in source code repositories [C]//Proceedings of the 12th Working Conference on Mining Software Repositories. Washington D. C., USA: IEEE Press, 2015: 396-400.

[7] 刘浩广,蔡绍洪. 信息熵及其随机性[J]. 贵州大学学报(自然科学版), 2007, 24(4): 350-351.

(下转第 129 页)

参考文献

[1] AGRAWAL A, KUMAR S, MISHRA A K. Implementation of novel approach for credit card fraud detection[C]// Proceedings of International Conference on Computing for Sustainable Global Development. Washington D. C., USA: IEEE Press, 2015: 8-11.

[2] BHATTACHARYYA S, JHA S, THARAKUNNEL K, et al. Data mining for credit card fraud: a comparative study[J]. Decision Support Systems, 2011, 50(3): 602-613.

[3] SEEJA K R, ZAREAPOOR M. FraudMiner: a novel credit card fraud detection model based on frequent itemset mining[J]. Scientific World Journal, 2014(2014): 1-10.

[4] NATH D M, JAMI S, JOG D. Credit card fraud detection using neural network[J]. International Journal of Students Research in Technology & Management, 2014, 2(2): 84-88.

[5] SRIVASTAVA A, KUNDU A, SURAL S, et al. Credit card fraud detection using hidden markov model[J]. IEEE Transactions on Dependable and Secure Computing, 2007, 5(1): 37-48.

[6] 冯　冲. 统计方法信息抽取中的若干关键技术研究[D]. 北京:中国科学技术大学, 2005.

[7] 朱莎莎,刘宗田,付剑锋,等. 基于条件随机场的中文时间短语识别[J]. 计算机工程, 2011, 37(15): 164-167.

[8] 古　勇,苏宏业,褚　健. 循环神经网络建模在非线性预测控制中的应用[J]. 控制与决策, 2000, 15(2): 254-256.

[9] QUAH J T S, SRIGANESH M. Real-time credit card fraud detection using computational intelligence[J]. Expert Systems with Applications, 2008, 35(4): 1721-1732.

[10] 谭小彬,王卫平,奚宏生,殷保群. 基于隐马尔可夫模型的异常检测[J]. 小型微型计算机系统, 2004, 25(8): 1546-1549.

[11] AGRAWAL R, GEHRKE J, GINOUULOS D, et al. Automatic subspace clustering of high dimensional data[J]. Data Mining and Knowledge Discovery, 2005, 11(1): 25-33.

[12] BHUSARI V, PATIL S. Application of hidden Markov model in credit card fraud detection[J]. International Journal of Distributed & Parallel Systems, 2011, 2(6): 33-36.

[13] CHEN Y, TU L. Density-based clustering for real-time stream data[C]//Proceedings of ACM Sigkdd International Conference on Knowledge Discovery & Data Mining. New York, USA: ACM Press, 2007: 133-142.

[14] ALI M H, SUNDUS A, QAISER W, et al. Applicative implementation of d-stream clustering algorithm for the real-time data of telecom sector[C]//Proceedings of International Conference on Computer Networks & Information Technology. Washington D. C., USA: IEEE Press, 2011: 293-297.

[15] CAVALIN P R, SABOURIN R C, SUEN Y, et al. Evaluation of incremental learning algorithms for HMM in the recognition of alphanumeric characters[J]. Pattern Recognition, 2009, 42(12): 3241-3253.

[16] FLOREZ-LARRAHONDO G, BRIDGES S, HANESAN E A. Incremental estimation of discrete hidden Markov models based on a new backward procedure[C]//Proceedings of IEEE National Conference on Artificial Intelligence. Washington D. C., USA: IEEE Press, 2005: 758-763.

[17] LEE W, STOLFO S J, MOK K W. A data mining framework for building intrusion detection models[C]// Proceedings of IEEE Symposium on Security and Privacy. Washington D. C., USA: IEEE Press, 1999: 120-132.

编辑　索书志

(上接第 121 页)

[8] Scanning data for entropy anomalies[EB/OL]. [2017-07-31]. http://blog. dkbza. org/2007/05/scanning-data-for-entropy-anomalies. html.

[9] PENG F, SCHUURMANS D. Combining naive Bayes and n-gram language models for text classification[C]// Proceedings of European Conference on Information Retrieval. Berlin, Germany: Springer, 2003: 335-350.

[10] NORVIG P. Natural language corpus data[EB/OL]. [2017-07-31]. http://www. norvig. com/ngrams/ch14. pdf.

[11] WANG Wei, SHIRLEY K. Breaking bad: detecting malicious domains using word segmentation[EB/OL]. [2017-07-31]. https://arxiv. org/abs/1506. 04111 2015.

[12] 邓爱萍. 程序代码相似度度量算法研究[J]. 计算机工程与设计, 2008, 29(17): 4636-4638.

[13] 石　野,黄龙和,车天阳,等. 基于语法树的程序相似度判定方法[J]. 吉林大学学报(信息科学版), 2014, 32(1): 95-100.

[14] COMANICIU D, MEER P. Mean shift: a robust approach toward feature space analysis[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2012, 24(5): 603-619.

[15] 张华伟,王明文,甘丽新. 基于随机森林的文本分类模型研究[J]. 山东大学学报(理学版), 2006, 41(3): 139-143.

[16] 张希翔,赵　欢. 基于随机森林的语音人格预测方法[J]. 计算机工程, 2017, 43(6): 253-258.

[17] KOHAVI R. A study of cross-validation and bootstrap for accuracy estimation and model selection[C]// Proceedings of International Joint Conference on Artificial Intelligence. New York, USA: ACM Press, 1995: 1137-1145.

[18] GOUTTE C, GAUSSIER E. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation[C]//Proceedings of European Conference on Information Retrieval. Berlin, Germany: Springer, 2005: 345-359.

编辑　顾逸斐