# 机器学习 – 动静结合检测恶意PDF文件

关键字：

机器学习 ，Python ，PDF ，特征提取

## 摘要ABSTRACT

在这个安全问题发展频繁的时代，恶意PDF文件在实践中仍然是对大多数计算机用户的真正威胁。其中一个原因是PDF的多平台性，并且其结构复杂多变，使恶意PDF文件的攻击更容易的被不法分子利用，尽管Adobe和其他供应商发布了一系列安全补丁，但许多用户仍然在其计算机上安装了易受攻击的客户端软件。

在本次实验中，我们利用 80K 样本集，依据恶意和良性PDF文件的结构属性的本质区别，总结了一些PDF恶意文件与正常文件之间的差异，针对这些差异，对现有的样本进行特征提取，利用机器学习来对文件进行训练和分类，经过多次测试和训练，分类效果均维持在 99%，目前分类效果良好。

## 1．简介INTRODUCTION

网络攻击者正在转向基于文件的恶意软件，因为用户聪明起来恶意电子邮件附件和网络链接，许多防病毒（AV）供应商[如建议39 ，50 ，54 ，57 ]。用户通常会被浏览器，电子邮件代理或AV产品更多地警告可执行文件的危险，而像PDF这样的文档由于受到静态文件的影响而受到的关注和审查要少得多，并且几乎没有什么危害。

但是，随着时间的推移，PDF规格已经改变。增加的脚本功能可以使文档以与可执行文件几乎相同的方式工作，包括连接到Internet的能力，运行进程以及与其他文件/程序进行交互。内容复杂性的增长为攻击者提供了更多的武器来发动强大的攻击，并且更灵活地隐藏恶意有效载荷（例如，加密，隐藏为图像，字体或Flash内容）并逃避检测。

一个maldoc通常利用其解释器中的一个或多个漏洞发起攻击。幸运的是（或者不幸的是），鉴于文档阅读器越来越复杂以及库/系统组件的依赖性越来越大，攻击者的攻击面越来越大。新的漏洞继续被发现，在2015年发布了137个CVE，在2016年仅有Adobe Acrobat Reader（AAR）。AAR的普及及其大型攻击面使其成为攻击者的首要攻击目标[ 25 ]，其次是浏览器和操作系统内核。在引入了类似Chrome的沙盒机制后[ 2 ]，在pwn2own竞赛中，单个漏洞的价值可能高达7万美元[ 21]]。收集的恶意软件样本显示，许多Adobe组件已被利用，包括元素解析器和解码器[ 37 ]，字体管理器[ 28 ]以及JavaScript引擎[ 22 ]。系统范围的依赖性，如图形库[ 23 ]也在攻击者的雷达。

随着PDF格式的普及，AAR的持续开发使得马尔科夫检测成为一个紧迫的问题，并且近年来已经提出了许多解决方案来检测具有恶意有效载荷的文档。这些技术可以分为两大类：静态和动态分析。

静态分析，或基于签名的检测[ 14 ，27 ，31 ，33 ，34 ，36 ，46 ，52 ，59 ]，解析的恶意内容的指示，如shellcode的或相似性与已知的恶意软件样本文档和搜索。在另一方面，动态分析，或基于执行的检测[ 45 ，48 ，58 ]，运行的部分或整个文档和迹线的恶意行为，如脆弱API调用或返回导向编程（ROP）。

PDF 文档的检测研究大多采用PDF 文档内容或结构为特征，利用随机森林、SVM、决策树等分类器构建PDF 检测器。例如，Charles 等人提出通过随机森林检测含恶意代码的PDF 文件技术PDFrate[47]，作者从PDF 文档元数据以及文档结构中提取了135 个特征，使用已标记特征的训练数据， 并采用10 倍的交叉验证，生成具有多个分类树的分类器，从待测PDF 文档提取特征，评估森林中的每个树，最后投票决定其分类。该方法初始训练过程计算开销较大，但一旦分类器构建完成，对待测的PDF 文档的分类速度很高。

然而，最近的作品突出了这些解决方案的一些共同的缺点。Carmony *等*。[ 11 ]表明，在这些解决方案中使用的PDF解析器可能已经过分简化了关于PDF规范的假设，导致恶意负载的不完整提取和分析失败。也已经证明，基于机器学习的检测有可能以原理和自动的方式被回避[ 35 ，

53 ，65 ]。另外，许多解决方案只关注JavaScript部分，而忽视了与其他PDF组件在发起攻击方面的协同作用。因此，尽管现代AV产品支持PDF漏洞检测，但即使后者仅对现有漏洞进行微小修改，也不能迅速适应新颖的模糊技术[ 55 ]。由于缺乏攻击程序和运行时间跟踪，AV产品还会出现提供零日攻击保护的问题。

## 2．相关工作

在机器学习领域已经有多篇关于对分类器的规避行为的攻击，主要集中在简单模型的垃圾邮件检测上[6,10,18]，Chinavle等人认为对手问题本质上是*概念漂移*，是考虑到数据分布随时间变化的机器学习领域[6]，然而，概念漂移解决方案假设数据分布的变化不是由于分类器本身造成的，而是由对手故意适应的。

Biggio等人以前已经研究过针对恶意软件分类器的逃避攻击。从分类模型[4]和Srndic et al。[28]。然而，这些研究假设攻击者只能插入新的特征，他们在特征空间中进行了逃避实验而没有产生实际的逃避PDF恶意软件。实际上，我们工作的实验表明，攻击者也可以在保留恶意的同时删除特征，而且我们的实验证实，由此产生的躲避变种通过在测试环境中的动态执行来保存恶意。

Srndic等人 展示了如何通过利用特征提取中的实现缺陷来避免PDFrate [29]。我们的方法不依赖目标分类器中的任何特定实现缺陷。相反，它利用分类器模型特征空间中的弱点，并采用随机方法以不同的方式处理样本。

Maiorca等人 提出了针对PDF恶意软件分类器的反向模仿攻击[20]。在反向模仿中，通过将恶意有效载荷插入到结构中，良性样本被操纵为恶意样本。这种攻击是基于结构特征的一类分类器通用的。但是，手工攻击仅适用于具有简单有效载荷的恶意软件。相反，我们基于GP的方法是自动的，并没有这个限制。

演化算法最近也被用来欺骗基于深度学习的计算机视觉模型[22]。相反，这项工作使用遗传编程，这是进化算法的一个重要分支，用于生成计算机程序等高度结构化的数据。

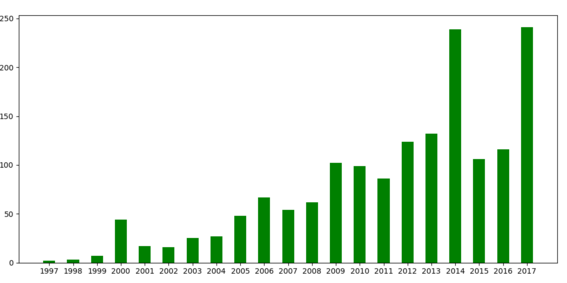
### 2.1 收集数据集

PDF文件被分类为良性或恶意，恶意进一步分为两类：normalPDF和MalPDF。

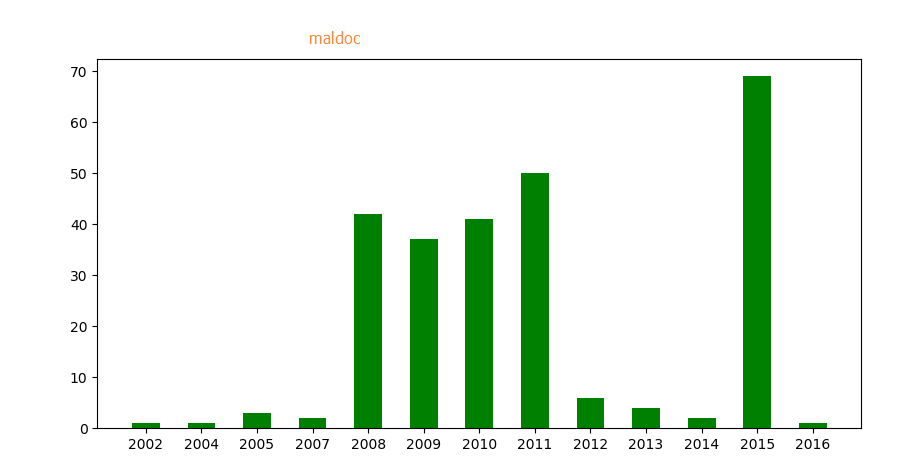
恶意 PDF 是指在正常的PDF 文件中嵌入恶意代码。传统的恶意PDF 检测方法有基于病毒检测[144]、基于签名的的检测方法[145]等，这些方法存在识别率不高、无法及时更新恶意代码等问题。机器学习技术为恶意PDF 检测提供了新方向。

最初收集恶意PDF文件 156035 个 ，截止时间是2017年4月。

以下是正常pdf文件样本的时间分布图时间从1997年到2017年



以下是恶意样本的时间分布图：由于恶意样本很多会对元数据进行加密，混淆等时间格式上的处理，所以对可以读到时间的样本就会减少，一下是可以读到时间的样本的时间分布图：



### 2.2数据清洗

对所有收集起来的数据进行初步筛选

目前有些收集的PDF文件，无法正常打开，缺少某些对象或结束符，我们先做了初步的筛选，以确保后面分析的是完整的可用的pdf文件。经过第一步的筛选之后 156035 个样本中只有83442个通过了此次筛选，之后被用做下一次的数据输入。

1. 数据集：

Normal sample number : 2026

Malware sample number : 83442

在做训练和推测的时候，随机抽取其中的部分做训练和测试做dataset

1. 操作：这些恶意样本文件是从virusshare 和蓝盾集群收集下载下来的，正常样本是通过爬虫抓取的 。

3. 了解PDF文件结构与恶意文件架构

PDF文件格式包含以下4个部分：

**文件头——指明了该文件所遵从的PDF规范的版本号，它出现在PDF文件的第一行。**

**文件体——又称对象集合，PDF文件的主要部分，由一系列对象组成。**

**交叉引用表——对对象进行随机存取而设立的一个间接对象的地址索引表。（实际以偏移+索引的方式储存对象地址，下文会提及）**

**文件尾——声明了交叉引用表的地址，即指明了文件体的根对象（Catalog），从而能够找到PDF文件中各个对象体的位置，达到随机访问。另外还保存了PDF文件的加密等安全信息。**

**PDF文件格式图示：**



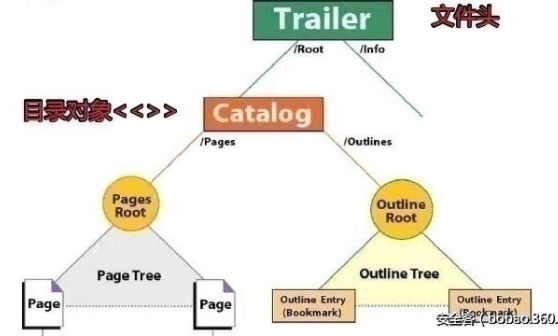
PDF文件的逻辑结构

本段主要介绍PDF文件体的读取方式。

作为一种结构化的文件格式，一个PDF文档是由一些称为“对象”的模块组成的。每个对象都有数字标号，这样的话可以这些对象就可以被其他的对象所引用。这些对象不需要按照顺序出现在PDF文档里面，出现的顺序可以是任意的，比如一个PDF文件有3页，第3页可以出现在第1页以前，对象按照顺序出现唯一的好处就是能够增加文件的可读性，对象的信息以偏移+索引的形式保存在交叉引用表内。

文件尾说明了根对象的对象号，并且说明交叉引用表的位置，通过对交叉引用表的查询可以找到目录对象(Catalog)。这个目录对象是该PDF文档的根对象，包含PDF文档的大纲(outline)和页面组对象（pages）引用。大纲对象是指PDF文件的书签树；页面组对象（pages）包含该文件的页面数，各个页面对象(page)的对象号。因此，PDF阅读器通常从文件末尾开始阅读PDF以提高效率。

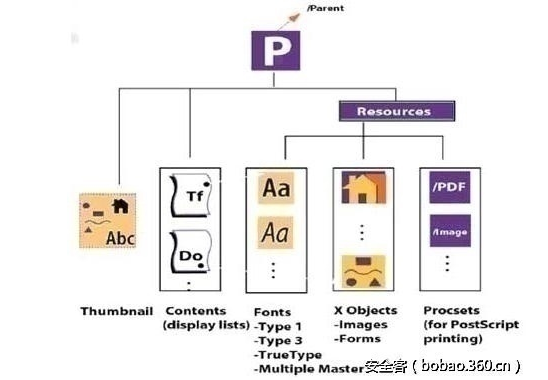
PDF的层级结构图示：



页面（page）对象为PDF中最重要的对象，包含如何显示该页面的信息，例如使用的字体，包含的内容（文字，图片等），页面的大小。里面的信息可以直接给出，当然里面的子项更多的是对其他对象的引用，真正的信息存放在其他对象里面。页面中包含的信息是包含在一个称为流（stream）的对象里，这个流的长度（字节数）必须直接给出或指向另外一个对象（包含一个整数值，表明这个流的长度）。

可见stream流对象我们恶代分析需要获取的重点。

页面信息图示：



理解了上面的内容之后，我们可以得出针对恶代分析的PDF文件的大致解析思路：

文件去除混淆

查找关键字

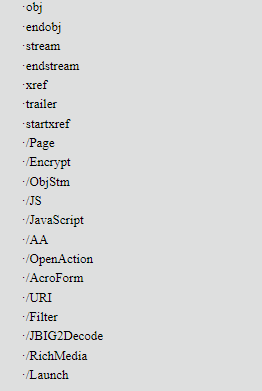
根据关键字获得可能存在恶意代码的stream 或者buffer

解码stream ，获取恶意代码

3．通过机器学习的方法，训练并得出预测模型

## 3．PDF文件结构和特征描述DATA & FEATURE DESCRIPTION

每一个PDF文件都包含有前7个字段，也有可能不包含strea和endstream。据说有一些ODF文件没有xref或则trailer，但是这种情况比较少见。如果一个PDF文件没有xref或者trailer关键字段，那么可以确定它不是恶意的PDF文件。



/xref 交叉引用表，描述每个间接对象的编号、版本和绝对的文件位置。PDF文档中的第一个索引必须从版本65535的0号对象开始，标识符/xref后面的第一个数字是第一个间接对象（即0号对象）的编号，第二个数字是/xref（交叉引用表）的大小。  
/Page指明PDF文件的页数，大多数恶意PDF文件仅仅只有一页  
/Encrypt指明PDF文件有数字水印或者是被加密过的。  
/ObjStm是object streams的数量。object streams是一个可以包含其他Object对象的数据流对象。  
/JS与/JavaScript指明PDF文件中含嵌有JavaScript代码。通常恶意的PDF文件都嵌套有JavaScript代码，这里一般都是利用JavaScript的解析漏洞或者使用JavaScript来实现堆喷射（heap spray），也有很多正常的PDF文件里会含有JavaScript代码  
/AA、/OpenAction和/AcroForm指明当查看PDF文件或者PDF的某页时会有动作随其执行，几乎所有嵌有JavaScript代码的恶意PDF文件都有自动执行JavaScript代码的动作(action)。如果一个PDF文件包含有/AA或/OpenAction自动执行动作的关键字段，而且含有JavaScript代码，那么这个PDF文件就极有可能是恶意的PDF文件  
/URI 如果你要在PDF文件中执行打开网页的动作就需要这个关键字段  
/Filter 一般为FlateDecode则是使用了Zlib压缩解压缩算法。  
/JBIG2Decode指明PDF文件使用了JBIG2压缩。虽然JBIG2压缩本身可能会有漏洞（CVE-2010-1297）。但/JBIG2Decode关键字段并不能说明PDF文件是否可疑  
/RichMedia Flash文件  
/Launch执行动作(action)数量

## 4 .系统设计

所提出的用于基于结构的检测恶意PDF文档的方法包括以下两个步骤，如图2所示：

1.     *提取结构特征。*作为基本的预处理步骤，PDF文档的内容被解析并转换成特殊的形式，*包 - 路径*，以一种定义明确的方式表征文档结构。

2.     *学习和分类。*检测过程是由恶意和良性的PDF doc

Benign file

Mal\_file

Choose 80% of training at random

Model

Classification

Unknown file file

Prediction

Feature Extraction

### 4.1   学习和分类

**4.     决策树**

## 5. 分类评估方法CLASSIFICATION METHODOLOGY

### 5.1 Feature Extraction特征提取

一般来说，动态分析引入了大量的开销和延迟，因为它本质上依赖于恶意代码的执行。这激发了基于静态分析的方法[19]以及静态和动态分析的结合[37]。这两种方法都着重于检测PDF文档中的恶意JavaScript内容，这些内容涉及大量的检测逐行下载的工作; 例如[27,31,10,6]。

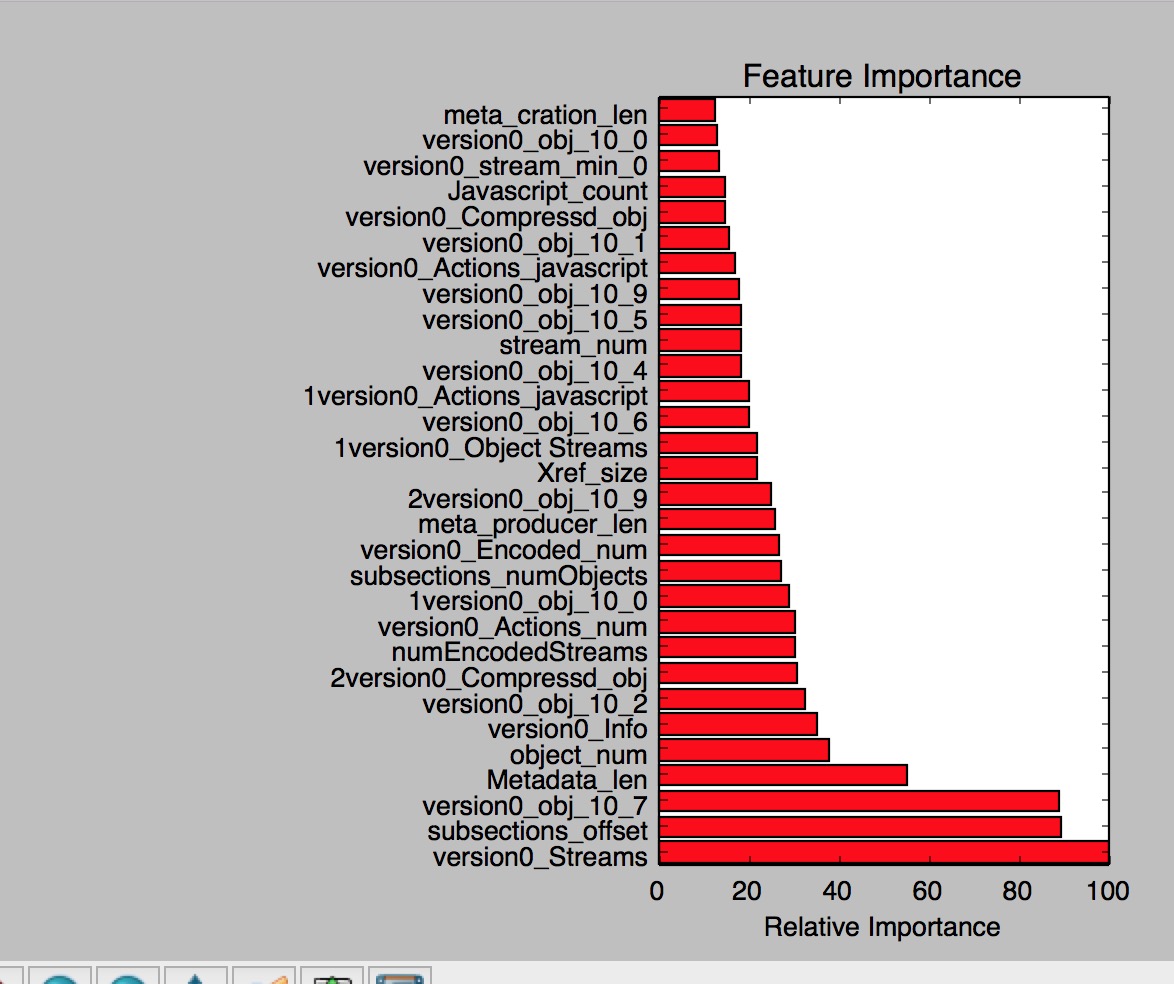
我们建议分析PDF文档的*结构属性*以区分恶意文件和良性文件。我们的方法不是寻找特定的恶意内容，而是根据其结构中恶意内容的副作用来评估文档。这种方法基于这样的直觉：由于PDF格式的复杂性，PDF文件的逻辑结构传达了大量文档的语义。因此，我们假定对结构性质的全面分析将揭示恶意和良性PDF文档之间的显着结构差异。在我们的主要贡献中，我们提出了一种新的PDF文档*结构*表示，表示为*结构路径*，可以作为各种数据分析方法自动处理的特征。

如以下这个图标，是每一个特征对应的含义:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | subsections\_numObjects | Xref\_subsections\_offset | |
| 2 | len\_URLs | 统计URL的数量 | |
| 3 | version0\_Actions\_JS | 版本0\_Actions\_JS | |
| 5 | XrefSection | XrefSection | |
| 16 | version0\_Events\_AA | 版本0\_Events\_AA | |
| 17 | version0\_Catalog | 版本0\_Catalog | |
| 18 | Xref\_errors | Xref\_errors | |
| 20 | Javascript\_count | Javascript 统计 | |
| 21 | Linearized | 是否线性化 | |
| 22 | version0\_elements | 版本0\_elements | |
| 23 | version0\_Elements\_EmbeddedFiles | 版本0\_Elements\_EmbeddedFiles | |
| 24 | version0\_obj\_average | 版本0\_obj\_average | |
| 25 | version0\_obj\_min | 版本0\_Object?Streams | |
| 26 | JS\_count | 逻辑树中的JS代码统计 | |
| 30 | version0\_Encoded\_num | 版本0\_Encoded\_num | |
| 42 | comments | 是否有注释 | |
| 43 | numEncodedStreams | 总的object 的数量 | |
| 45 | meta\_cration\_len | metadata cration | |
| 47 | Metadata | metadata len | |
| 49 | version0\_Decoding Errors | 版本0\_Decoding?Errors | |
| 57 | meta\_creator\_len | metadata creator | |
| 58 | meta\_producer\_len | metadata producer | |
| 59 | version0\_Info | 版本0\_Info | |
| 61 | stream\_num | Xref\_subsections\_entries | |
| 63 | Encrypted | 是否加密 |  |
| 64 | version0\_Actions\_num | 版本0\_Actions\_num | |
| 66 | Binary | 是否是二进制 | |
| 67 | subsections\_size | 是否有 trailer | |
| 68 | header\_offset | 文件头偏移 | |
| 69 | trailer\_num | 逻辑树的大小 | |
| 70 | version0\_Events\_Names | 版本0\_Events\_Names | |
| 71 | version0\_Compressd\_obj | 版本0\_Compressd\_obj | |
| 76 | font\_count | 字体统计数量 | |
| 77 | Xref\_bytesPerFisId | Xref\_bytesPerFisId | |
| 78 | version0\_Events\_num | 版本0\_Events\_num | |
| 80 | Versions\_num | Versions\_num | |
| 81 | subsections\_entries | Xref\_subsections\_errors | |
| 82 | version0\_stream\_min\_0 | 版本0\_stream\_min\_1 | |
| 83 | version0\_stream\_min\_1 | 版本0\_Streams | |
| 91 | error | 是否有错误 | |
| 92 | Xref\_offset | Xref\_offset | |
| 93 | JS\_MODULE | 有没有JS模块 | |
| 94 | version-1 | 版本前部分int | |
| 103 | subsections\_errors | Xref\_subsections\_firstObject | |
| 106 | object\_num | 总的Stream 的数量 | |
| 107 | subsections\_firstObject | Xref\_subsections\_numfirstObject | |
| 108 | file\_size | 文件大小 |  |
| 114 | version0\_Vulns | 版本0\_Xref?Streams | |
| 115 | version0\_Object Streams | 版本0\_Objects\_JS\_num | |
| 117 | subsections\_offset | Xref\_subsections\_size | |
| 119 | Xref\_stream | Xref\_stream | |
| 122 | version0\_Objects\_JS\_num | 版本0\_stream\_min\_0 | |
| 125 | meta\_author\_len | metadata author len 元数据作者 | |
| 126 | update | 是否有更新 | |
| 127 | version0\_Xref Streams | 版本0obj\_min | |
| 129 | version-2 | 版本后部分int | |
| 130 | version0\_Streams | 版本0\_Vulns | |
| 131 | Metadata\_len | 编译Stream 的数量 | |
| 132 | Xref\_size | Xref\_size | |
| 133 | version0\_Actions\_javascript | 版本0\_Actions\_javascript | |

由上图可以看出，我们是通过分析PDF 的结构，根据一些恶意PDF特有的一些特征进行提取，静态分析

以下是前30个的分类特征图表：



### 5.2 算法选择

**随机森林(Random Forest,简称RF)**

输入为样本集D={(x,y1),(x2,y2),...(xm,ym)}D={(x,y1),(x2,y2),...(xm,ym)}，弱分类器迭代次数T。

　　　　输出为最终的强分类器f(x)f(x)

　　　　1）对于t=1,2...,T:

　　　　　　a)对训练集进行第t次随机采样，共采集m次，得到包含m个样本的采样集DmDm

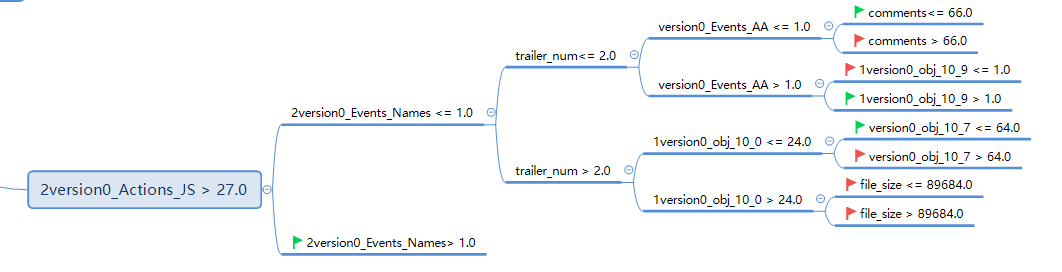
　　　　　　b)用采样集DmDm训练第m个决策树模型Gm(x)Gm(x)，在训练决策树模型的节点的时候， 在节点上所有的样本特征中选择一部分样本特征， 在这些随机选择的部分样本特征中选择一个最优的特征来做决策树的左右子树划分

　　　　2) 如果是分类算法预测，则T个弱学习器投出最多票数的类别或者类别之一为最终类别。如果是回归算法，T个弱学习器得到的回归结果进行算术平均得到的值为最终的模型输出。

如下是在NOTEBOOK上用scala推算时生成的树





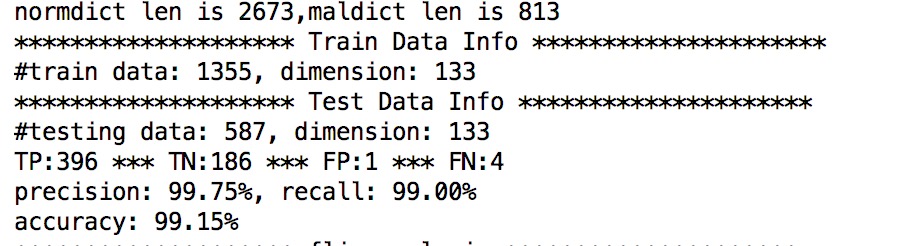


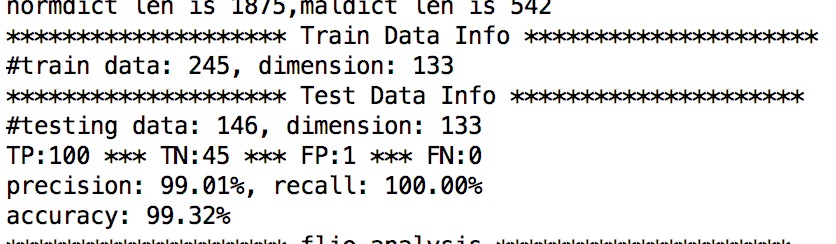
### 5.3动态检测

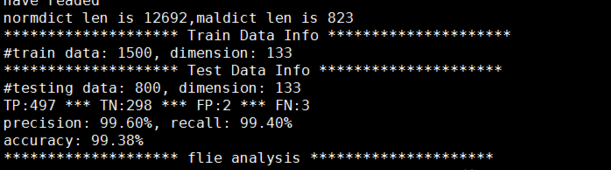
## 5. PERFORMANCE EVALUATION性能评估

### 5.1 Classification & Detection Performance

* 模型识别率： 平均模型识别率均在 98.00% ，使用特征133个经过多次测试
* 以下是在样本中随机抽取2300个样本做出的一个预测结果截图：模型识别能力达 >= 99.00%







## 6.总结

## 7.前景展望

## 8.参考文献

[2] Adobe Systems Inc. Introducing Adobe Reader Protected

Mode, 2010. http://blogs.adobe.com/security/2010/

07/introducing-adobe-reader-protected-mode.html.