Javascript ?

静态的

# 题目：

# 基于机器学习的PDF文件检测与分类器逃逸

## 摘要

如今信息发展非常的迅速，基于恶意文档（如pdf）的检测越来越重要，文档类型的攻击具有针对性，并且更容易攻击成功。原因之一在于结构与形式的多样，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，攻击者手段之丰富（如通过电子邮件的附件传播，钓鱼网站加密混淆等 ）。其次网络攻击者正在转向基于文件的恶意软件，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。  
此研究项目的目标旨在为工业界和学术界提供一个基于AI的文档（PDF，word等）分类器，我们收集了200 000个样本进行检测，这些样本大部分是通过VirusShare 下载下来的，一部分来自于contagio，一部分通过爬虫收集了百度和搜狗的正常文本，类型多样和丰富，通过特征提取的特征计算，可以使其检测率达到99.81%，误报率0.16%，当个文件检测时间为毫秒级别。通过对文档领域特征的挖掘，并结合数据科学中的复杂建模，模型成功被构建。此模型可广泛应用于终端安全产品，又或者在邮件服务器，均是非常有意义的应用场景，AI文档分类器不仅限于pdf，如可扩展到word，execl，ppt等有结构的文档

## 简介

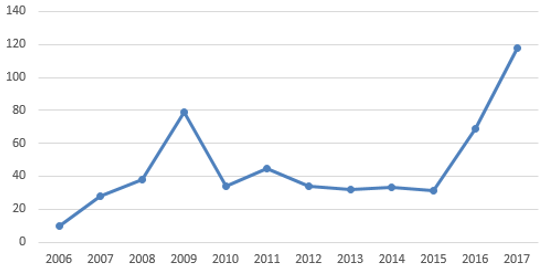
随着时间的推移，PDF规格和样式都在改变，变得丰富。增加的脚本功能可以使文档以与可执行文件几乎相同的方式工作，包括连接到Internet的能力，运行进程以及与其他文件/程序进行交互。内容复杂性的增长为攻击者提供了更多的武器来发动强大的攻击，并且更灵活地隐藏恶意有效载荷（例如，加密，隐藏为图像，字体或Flash内容）并逃避检测。

由于企业和个人用户系统未采用最新的浏览器插件（例如，Adobe的 Flash Player 和 Acrobat Reader）修补程序进行更新，这些攻击得以取得成功。

在2012年，开创性的漏洞攻击工具Blackhole专门针对Java，PDF和Flash文件。2013年，通过网络传输的不可执行恶意软件主要是针对Adobe Reader和Microsoft Office应用程序的PDF和Flash文件。近期Flash已经广泛部署恶意广告，即通过广告网络在合法网站上放置恶意软件。甚至一些最著名的网站也成为这种攻击的受害者

在2014年发现的24个0day中[6]，有16个是针对Adobe Reader和Flash Player的）

表1 Adobe Reader 在CVE中的增长趋势



无处不在的勒索软件（可能可以放在相关工作中）

勒索软件是一种恶意软件，感染后它会阻止你访问文件系统，以此来要挟你支付赎金，这类的软件通常要求使用比特币支付，要是你在一定的时间内没有进行支付，它会自动将赎金加倍。而且在你支付之前，这个循环会无限滚雪球下去。这种攻击的强度和频率会持续增加，而它给受害者造成的损失也会一路狂飙。

部分数据统计“勒索软件受害者的数量在上升，在2015年4月到2016年3月之间有大概718536位受害者，这相当于2014-2015年同期的5.5倍。”

中国依然是垃圾邮件的重灾区

在2017年的Symantec 的年度互联网安全威胁报告（Internet Security Threat Report，简称ISTR），中国在Malicious activity by source 总排名中名列前茅，

According to Symantec 2017 Internet Security Threat Report，

其中在Spam zombies的统计中更是远超其他国家，排在了最前面的位置

2016年，Cymantec观察到大量通过恶意Office宏（W97M.Downloader和变体）和JavaScript下载程序文件（JS.Downloader和变体）分发勒索软件和网上银行威胁的电子邮件活动。 在他们之间，他们在2016年终端上的检测数量达到了700万，并且主宰了网络犯罪威胁的形势，特别是在当年下半年

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F击面使其成为攻击者的首要攻击目标，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，在pwn2own竞赛中，单个漏洞的价值可能高达7万美元。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，字体管理器]以及JavaScript引擎。系统范围的依赖性，如图形库[ 23 ]也在攻击者的雷达

In This paper ,我们通过对不可执行的文档PDF的恶意行为进行研究，，我们提供了一个案例研究的结果，我们在一个真正的基于学习的系统上进行了PDF RATE，一个用于检测PDF恶意软件的在线服务[23]。对于任何提交的PDF文件，PDF RATE 提供了对其恶意的概率估计。我们的研究解决了攻击者试图通过修改提交的PDF文件来逃避检测的情况，以便其恶意功能保持完好，但PDF RATE 返回的概率得分降低。

我们已经分析了PDF RATE 的作者提出和评估的防御策略，虽然我们不知道它是否部署在在线系统中。文献[23]的攻击方案假定攻击者利用一小部分信息特征。结果表明，通过将一小部分预期的攻击数据包含到训练集中，可以有效地阻止这种攻击。我们重建了这一攻击，并验证了原有防御战略的有效性。然而，这种主动防御最终只能有效抵抗攻击的“紧张”。每当执行的攻击与预期攻击不符时，主动防御的效果基本消失，检测精度降到10％以下

本文的主要贡献如下：

* 目前数据集从业界通用的万级别 到 达到 二十万 级别以上，其中恶意样本173036个，正常样本28332个。
* 定义新的特征加入到分类器中，使其可以有效的区分PDF 恶意与良性的样本(特征)
* （测试鲁棒性）使用hidost2016[?]，并使用新的病毒变种（malware variants，请介绍病毒变种的定义）来测试模型的鲁棒性（第一次使用业界领先的？？方法去测试模型的鲁棒性）两个图标支撑
* 成功实现了一种在各种情况下针对自己生成的分类模型进行回避攻击（分类器逃逸攻击）（有一个3.1版本支撑）
* 重新训练之前的model\_2，生成新的Model,使其在分类器抗逃逸与变异病毒（病毒变种）的检测效率（检测率）有所提高

1，数据集，特征集，模型 –》我们基本解决了 PDF Malware Detection （ML）

2，逃逸

## 相关工作(2018/05/22)

如今检测PDF文件的方法有很多，主要的方法可以分为两类，动态检测和静态检测，纵然这有一定程度上的重合。动态方法需要把文档放在某一个特定探针环境（instrumented environment）中打开运行；而静态方法则无需打开运行仅通过静态特征即可完成检测。

一些核心想法促成了动态分析方法的兴起。最开始的工作是基于模拟的方法，此方法把可疑包（suspicious payload）使用抽象包执行（abstract payload execution）[36]或使用软件模拟[1，26]。然而，软件模拟并不能有效覆盖全部指令集。正因如此，有的文件就会逃逸检测。为了解决上述问题和增强可扩展性，最近提出的系统SHELLOS在可控shellcode执行的过程中[34]中使用硬件虚拟化技术以取代模拟方法。在一些常用的操作系统核心上，SHELLOS能有效地检测出shellcode（此shellcode在被程序任意分配的缓存之中）。然而，这么好的效果有一定的代价。尽管SHELLOS在检测网络层攻击时显示出优秀的吞吐量，但是检测恶意文档的时却遭受严重的延迟（在秒级别）。这种延迟造成的原因是检测位于内存缓存（memory buffers）中，此类检测需要程序首先开辟内存缓存，才能进行检测。

另外一种动态分析的方法聚焦于检测在运行JavaScript时的恶意行为。JSAND使用10个仔细提取的启发性特征去训练良性的JavaScript模型，然后使用此模型去检测与此模型基线有较大偏移的攻击[9]。一个相似的方法也被成功运用于ActionScript3恶意软件的检测当中[24]。CUJO建立在专门针对JavaScript的沙箱上，且可自动学习事件序列模型（models of sequences of events），这些模型均对JavaScript的解析器产生影响[31]。专门的Java Script动态分析方法与shellcode检测方法相比，有很大的性能提升（单个文件的检测时间控制在数百毫秒内），同时也保持了高准确度和低误报率。

PJScan是第一个，成功实现基于JavaScript内容的PDF恶意软件检测[19]。为了提高效率，PJScan的JS提取器只搜索那些PDF标准预先指定好的位置。不幸的是，这种提取方法会在以下情况下有局限，攻击者可以将JS代码放在其他位置，通过PDF JavaScript API访问，并使用eval()函数调用来获取，以致来逃逸检测。

另外Maiorca等人提出的系统MALWARE SLAYER[23]，基于模式识别。这里的模式是指使用PDFID工具从PDF文档中提取的文本关键字。它在真实PDF数据集上展现出高准确度和低误报率。这种方法被PDFID自身的函数所局限，且不能处理多版本（multiple revision numbers）和在对象流（object stream）中的隐藏对象（object hidden）。

传统的恶意PDF 检测方法有基于病毒检测、基于签名的的检测方法[14]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，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，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，我们同时也对这135个特征进行了提取，利用随机森林的算法对文档进行预测分类。

[14]Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto. Lux0R: Detection of Malicious PDF-embedded JavaScript Code through Discriminant Analysis of API References. In Proceedings of the Artificial Intelligent and Security Workshop (AISec), 2014.

PDFrate是一个相对成熟基于机器学习的静态PDF分类器。此分类器使用简单的PDF附带信息（meta info）和字节层面（byte-level）的文件格式。此PDFrate在大数据集中有出色的分类表现[33]。然而，此分类器不能提取对象流（object streams），在检测时这个特征就不能加入到训练样本中。

Maiorca等人。也研究了对PDFrate和其他PDF文档分类器的逃逸[26]，[27]。他们提出 反向模仿技术。为了使内容看似良性（如Mimicus所做的那样），他们将恶意内容嵌入到良性PDF中，而不是将内容添加到恶意文档中，而是尽可能少地修改。反向模仿攻击实施针对PDFrate的独立逃避方法, Maiorca等人提出了三种不同的逃避方案。在EXEembed方案中，恶意可执行文件被植入到现有的良性PDF文档中。打开文档时会执行恶意软件。这些文档利用CVE-2010-1240。在PDFembed方案中，恶意PDF被嵌入到良性PDF中。这些嵌入式文档在文档打开时自动呈现。为了评估，Maiorca等人 将利用CVE-2009-0927的文档嵌入到现有的良性PDF文档中。最后，在JSinject场景中，恶意JavaScript（与PDFembed嵌入式文档中使用的相同）直接注入根良性文档。

另外两个须提及的贡献是把静态和动态分析技术结合起来。MDScan[37]对PDF文件进行静态分析，目标是提取所有的JS代码簇（chunk），这些代码簇可以作为执行的入口（entry point）。直到现今为止，一个特定用途的解析器（parser）被MDScan开发，这个解析器被用于抓取在文件里面的其他信息，这些信息包括被交叉引用图表（cross-reference table）所忽略的对象以及潜在的恶意对象。被抓取的代码会在JavaScript引擎中被执行（此引擎会模拟Acrobat Reader引擎）。在控制执行中（controlled execution），所有的内存缓存都会被一个工具所检测，此工具基于二进制模拟（binary emulation NEMU）以用于shellcode检测。

在ZOZZLE[10]一文中，静态与动态部件的角色发生了变化。ZOZZLE的动态部件会在运行前提取JavaScript的片段，这些JavaScript源于IE浏览器中的JavaScript引擎，此引擎能自然而然地解决JavaSript代码混淆（obfuscation）的问题。ZOZZLE的静态分析部分使用Bayesian分类器，此分类器建立在对检测到的JavaScript源代码进行混合分析（syntactic analysis）的基础上。

对相关工作的比较表明，在当前用于文档恶意软件检测的静态和动态系统间有明显的权衡（trade-off）。纵然动态分析系统表现出高准确度和低误报率，这种优势是以高延迟、明显的性能瓶颈和需要特定的受控制系统（instrumented environments）为代价。在本文中提出的新方法，旨在从静态方法填补空隙。此技术通过从静态方法开始，增强检测性能同时保持系统设计的简约和计算的高效。为了达到这样的目标，我们使用了一个现成的解析器，构筑关于“Comprehensive Static Analysis of PDF”的方法论。更进一步地，我们对样本潜在的逃逸策略有特别的关注，且用实验去评估所提及的方法对特定攻击策略的健壮性。

相关工作：

动态分析与静态分析

基于签名与机器学习的问题

## PDF文件分析

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

### 3.1 PDF文件结构分析

*可移植文档格式*（PDF）是一个开放标准，发布为ISO 32000-1：2008 [ [9](https://link.springer.com/article/10.1186/s13635-016-0045-0#CR9) ]。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：**



**参照PE论文 PE Header Format**

本段主要介绍PDF文件体的读取方式。

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

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：

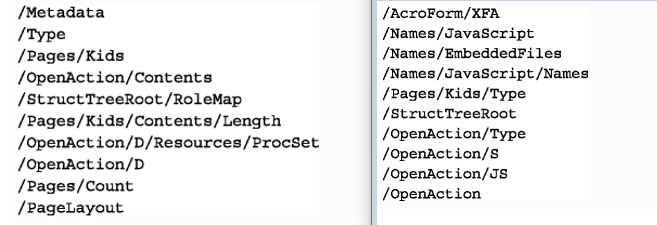
（**请删除以上图表或重新画**）

### 3.2 PDF特征选择

理解了上面的内容之后，我们可以得出针对恶意PDF文件分析的大致解析思路：

据调查显示，根据PDF结构路径在文件中存在最能表明文件是良性的，或者缺少文件结构等表明文件是恶意的。例如，恶意文件不太可能包含元数据以最小化文件大小，打开文档时不会跳转到文档中的页面，并且格式不正确，因此它们缺少路径，例如/ Type和/ Pages　／count

以下列表显示了来自真实世界良性PDF文件和恶意PDF文件的示例性结构路径：（左边是良性的，右边是恶意的 ？？）



* /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

Table 1: Mean Values between Benign & Malware Samples

|  |  |  |
| --- | --- | --- |
| Features | Benign Files | Malware Files |
| **count\_acroform\_obs** | **700** | **1400** |
| count\_box\_a4 | 12001 | 200 |
| count\_box\_legal | 395040 | 0 |
| **count\_box\_letter** | **7291529** | **866773** |
| count\_box\_other | 32.184318 | 1.749550 |
| count\_box\_overlap | 1000 | 0 |
| count\_endobj | 95.800680 | 9.685137 |
| count\_endstream | 30.436044 | 3.785357 |
| **count\_font** | 14.646365 | 0.551510 |
| count\_image\_large | 110711 | 400 |
| count\_image\_med | 465247 | 6401 |
| **count\_image\_small** | **915892** | **12002** |
| count\_image\_total | 36.56366 | 0.30006 |
| count\_image\_xlarge | 300 | 0 |
| count\_image\_xsmall | 21.64216 | 0.11202 |
| count\_js | 0.714071 | 1.012402 |
| **count\_obj** | 100.969997 | 12.014803 |
| count\_objstm | 1.572057 | 0.153831 |
| count\_page\_obs | 8001 | 16003 |
| count\_startxref | 1.537254 | 0.989598 |
| count\_stream | 30.587159 | 3.807161 |
| title\_len | 12.682468 | 1.723145 |
| creator\_len | 12.211721 | 3.611322 |

Count\_OBJ

Count\_font

Count\_js

## 数据集与数据清理

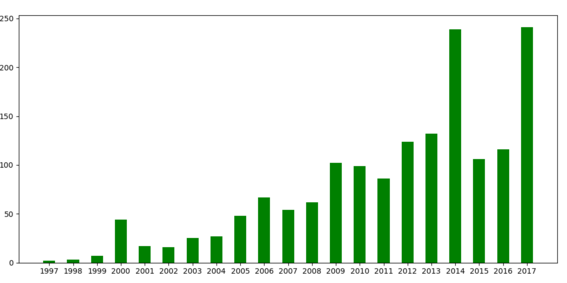
### 数据集

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

最初收集恶意PDF文件 156035 个 ，截止时间是2017年4月。

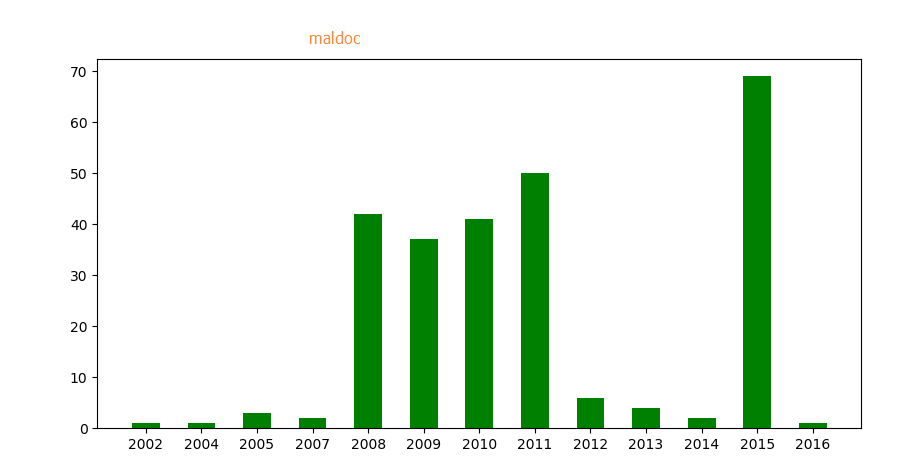
有三个数据集参与了PDF RATE 的创建和评估。已经对三种模型进行了训练，分别使用这些模型来评估用户提交的新数据。

三个数据集中的两个用于[23]中的PDF RATE 实验性评估：Contagio 。该Contagio数据集贡献的恶意软件研究的恶意和良性PDF文件的集合，可供下载。训练PDF RATE 是在Contagio数据集的子样本上进行的，该样本包含5,000个良性文件和5,000个恶意文件。经过训练的分类器在操作 数据库上进行了评估，包括在“大型大学校园”收集的100,000个PDF文件。以下是正常pdf文件样本的时间分布图时间从1997年到2017年



（**图上加入趋势线，加上下标，图上的数据是如何收集而来，并且做预处理的**）

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：



（重新选取更大的数据集 去 画图）

### 数据筛选

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。

之后我们针对第一次未分析到PDF文件进行重新复检，发现PDF利用部分结构不全导致分类器无法识别，可是依然有恶意性质保留，进而逃逸分类器的解析器，现在我们就在针对第一批逃逸的对象不全的文件进行进一步解析，重新测试并且利用这PDF文件重新生成了一批新的变异病毒，来训练测试分类器使其对新的逃逸病毒具有很好免疫。

最后数据集从业界通用的万级别 到 达到十万级别以上，包含有两万的正常样本和16万的恶意样本

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 。

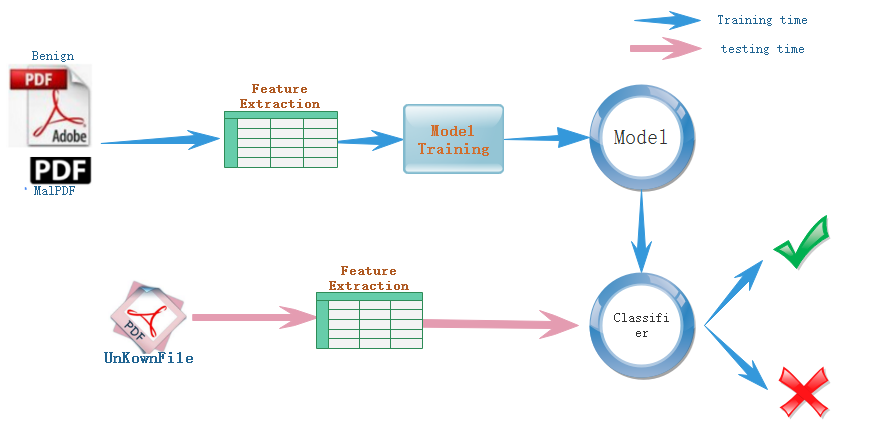
## 分类评估方法

### 训练分类模型构建

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，如下图所示：

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



**参看PE与APK文章，图6分**

在这项工作中，我们关注分类器，这是一种特殊类型的学习系统，它将新数据分为两个或多个预定义的类别。分类器通常通过计算一些数字或概率分数并将其与固定阈值进行比较来做出预测。旨在操纵分类器的对手的目标是将其混淆 为提供错误的分类。对于二元分类问题，错误分类被称为false positives和错误否定false negatives。从对抗的角度来看，有关基于学习的系统的信息越多，系统成功的机会就越大。

构成每个基于学习的分类器系统的基本组件是：

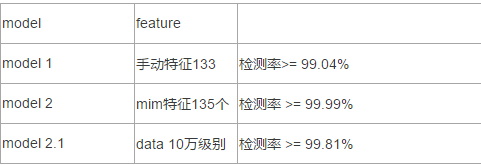
•    分类器使用的一组特征

•    训练数据集用于分类器训练

•    分类算法及其参数。

因此，为了对手的利益，最大化他们对目标分类器组件的知识。例如，对手*一个*谁知道某个分类的功能集和训练数据集有逃避它比对手的机会较高*乙*谁只知道功能集。从这个意义上说，两个对手*A*和*B*在不同的*逃避场景下*运作。逃避场景是从对手的角度来看逃避的问题。它以结构化的方式描述对手可用的分类器系统信息：概述对手对于特征集合，训练数据集合和分类算法的误报率是低还是高。

如下图是更新后模型的一个改进与与更新，在特征与数据集上有所改进，具体的就是Model 2 在特征选取上更为多，由之前手动提取的133个特征，增加到135个，并且也是在学术界中比较认可的135个特征集来训练。Model2 是一个中间测试的过度模型，Model2.1训练数据集为十万级别，并且所有测试样本都是病毒的变种样本，模型检测率为99.81%



### 算法选择

PDF RATE 采用随机森林[31]，一种集合学习方法，包括独立训练的决策树的数量*t RF*。在训练步骤中，使用CART方法学习每棵树，但只使用可用训练样本的一个子集。通过从训练数据中随机抽样一定数量的树来为每棵树生成一个不同的子集，用替换 - 称为引导聚合或装袋的过程。当新的决策节点被添加到树中时，只考虑*f个RF*特征的随机选择的子集，其中*f RF*小于功能的总数。通过在给定的新数据点上的所有决策树中进行多数投票作出决定。随机森林以其卓越的泛化能力和对数据噪声的稳健性而闻名。PDF RATE 使用Leo Breiman's和Adele Cutler的原始随机森林实现的R端口，作为软件包RANDOM F OREST。*t RF*和*f RF*分别是称为*ntree*和*mtry*的RANDOM F OREST的 参数。值*ntree* = 1000 和*mtry* = 43由PDF RATE使用。

所有这三个分类的PDF部署率，即接受了有关的那些*Contagio*， *GMU*和*社区*数据集，生产他们的结果是他们的决策函数在区间输出，即真正的价值[0 *，* 1] 表示的百分比将提交的文件标记为恶意的决定。在[23]中没有给出阈值来确定一个文件应该被认为是恶意的。请注意，通过提供此百分比值而不是二元决策PDF RATE揭示了更多关于分类引擎的信息，而不是决策所必需的信息，从而使对手能够在制定逃避方法时作出更明智的决定。

**随机森林(Random Forest,简称RF)**

（与其用正式的语言去定义随机森林，用非正式的语言可能会更好）。Random forest -> large number of decision trees -> for each iteration, decision trees can be adjust -> At last, the result of the RF is derived from the majority of the results of the decision trees

(增加与SVM的对比，如有可能，阐述一下算法见的比较是蛮好的)

(考虑加入 random forest中的grid search模型调参，参看PE论文 **Figure 3：Grid Search for Model Parameters**)

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:

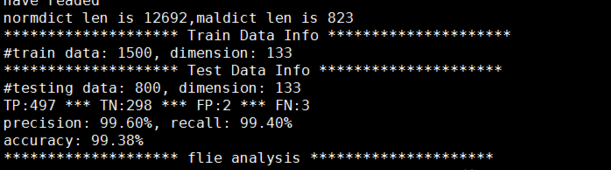
　　　　　　a)对训练集进行第t次随机采样，共采集m次，得到包含m个样本的采样集DmDm

　　　　　　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。

### 分类结果

* 模型识别率： 平均模型识别率均在 99.30% ，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%



如下是在NOTEBOOK上用scala推算时生成的树，IPython notebook目前已经成为用Python做教学、计算、科研的一个重要工具。该工具既能动态驱动，还具备可重复生成的特点。我们在试验中也使用这个工具对我们提取的数据进行测试，使用数据集是十万级别， 并且分类效果也达到了 99.30%

（在论文后面添加一个附录，特征以及名称）

在数据集为10万级别时，分类准确度达到99.30%。

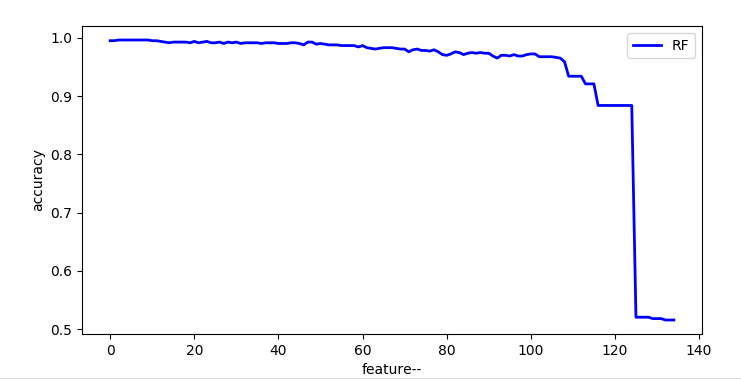


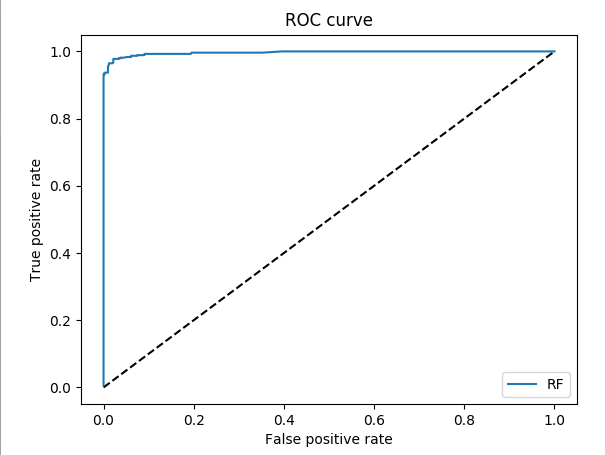
如下图所示，决策树比其他算法优胜的地方在于她良好的可解析行，如下图所示。

（**论述一下训练的样本数，整棵树的规模，如深度，分支，最重要的特征为。。。**）下图是分类器在对特征进行分类的过程中生成的一颗树的其中一个枝子，其中就是以Version\_Action\_JA 的这个特征作为结点的，这个特征是区分正常与恶意样本的一个关键特征，因为很多恶意文件的恶意代码都会嵌入到 Action\_JS 这个对象当中，而在正常样本很少会有使用这个对象。下一个节点就是Version\_Streams,统计版本中流的数目，恶意文件的流对象一般会比正常文件的流对象要多一些，当然这个只是作为一个枝子，在后面还要进行合并判断，比如版本号，Metadata 的长度，Javascript的数量与分析，等等，多个特征合并计算权值后投票对文件进行判断。



### 模型评估：





## 讨论

### 分类器逃逸分析

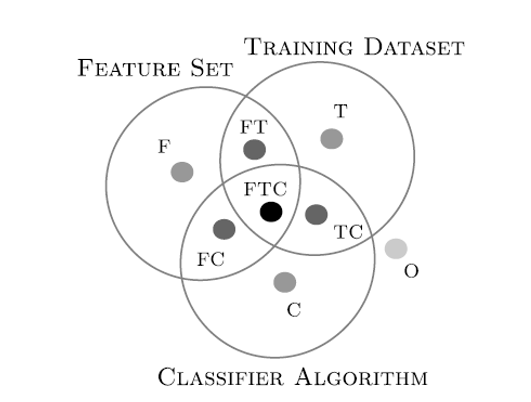
为了系统地探索针对分类器系统的逃避攻击，基于知识分子拥有的关于分类器系统三个组成部分的知识量，提出了图1所示的逃避场景分类。

● 在情况F中，只有特征集可用于敌手，在不同程度上。攻击者可能会意识到某些或全部特征，错误地将过时的特征视为正在使用，能够读取子集或全部特征或者能够以不同程度修改某些或全部特征。然而，为了能够修改样本并进行规避，需要操纵足够的特征子集。

● 在FT情况下，除了已知的特征之外，攻击者还可以利用目标分类器训练数据集的知识。数据集可能完全或部分泄漏，从而在生成成功的攻击样本的过程中实现更准确的决策。

● 在场景FC中，攻击者知道特征集以及关于分类器的一些细节，例如类型，参数或具体实现。根本没有关于训练数据集的信息而没有替代数据集的对手几乎不知道分类器的优点。使用替代数据集，他们可以训练正确类型的替代分类器，但是这种近似的准确性取决于收集的数据的质量。这种攻击也可以离线执行，类似于基于替代分类器的其他攻击。

● FTC如果知道所有三个分类器组件的细节，那么对手就有最大的机会躲避目标分类器。在这种情况下，他可以在线下完全重现在线分类器，只有在找到足够好的规避样本时才提交攻击结果。离线模仿攻击或离线分类器特定攻击击败离线分类器也具有很强的击败在线分类器的可能性。



这个的分类包含8个逃避场景。他们的名字描述了对手可用的信息。如果对应于分类器组件特征集的字母F，T或C中的任何一个分别存在于场景的名称中，则训练数据集和分类器算法分别存在于对手所具有的给定分类器组件的知识水平这种情况很高，否则低。名为O的情景指的是敌手对所有三个分类器组件都知之甚少的情况。

### 更新分类模型

### 攻击结果

攻击样本选取：

攻击样本的选取只要针对Model逃逸的样本做进一步的筛选，有很大一部分样本，在结构上会有一些省略或对象不完整，导致解析器无法解析，就无法提取到完整的特征，因为这个原因分类器就无法对此样本进行近一步的分类判断，在很大程度上就逃逸了分类器，其他的的一些逃逸方法还会有编译器混淆，加密，多层包含，等等，

在Model1中最开始有接近一半的样本（7万多的样本逃逸分类器）没有被解析到，进而没有参与训练分类，可是样本依然有恶意的代码，保留有其恶意的属性，在Model2 训练的时候，就在这7万多的样本中选取逃逸较多的那些，多次测试后使用FP的样本，使用Nedim Srndic在SP会议中的基因编程技术，生成新的变种来测试分类器的鲁棒性。我们最后使用2038个逃逸样本来测试Model，然后再训练一个新的Model2.1, 如下图，很明显的，在经过编译后，Model的攻击相对来是成功的，几乎80左右的样本逃逸。

在本次攻击的的几个案例中，我们主要正对上面说到的4种情况进行病毒变种与分类器逃逸攻击。

**对进行评估的2038有一定的描述**

**FC一行攻击得有效**

**为什么FC那么有效？为什么F就一般？**

**找一个测试集，彰显变异的效果**

**建立一个新的模型，他采用的特征都是不容易被逃逸的。**

**从以上表中看出了TODOs**

**（参考 Table6，7 and figure6 from PE论文）**

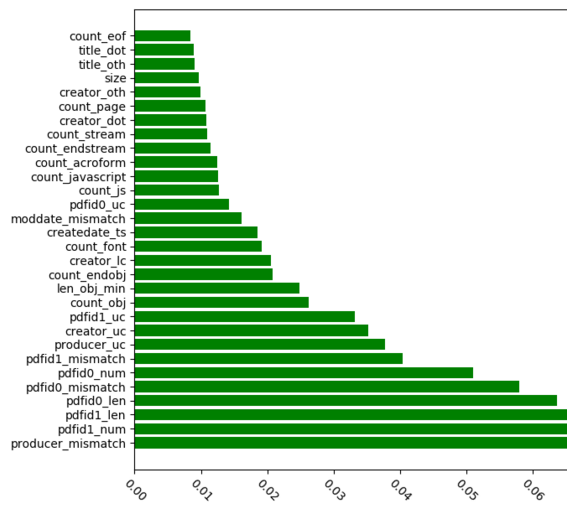


Figure 4：Important Analysis of Features

**加数据分析**

### 预防措施

在我们上一个实验中，我们已经研究了[23]中提出的防御机制对我们的逃避技术的鲁棒性。为了设定基线，我们使用在*Contagio*数据集上训练过的随机森林分类器，完全按照Smutz和Stavrou提出的模拟攻击和防御技术（参见第IV-D节）重现模仿攻击和防御技术。我们的分类器按照最重要的顺序排列了以下30 个特征：



模仿攻击唯一可以想象的特征是具体特征的经验支持，即攻击改变了特定特征的文件的百分比。有几乎每次攻击都会改变的45个特征。剩下的23个特征很少被修改，很可能是由于相反的变化方向（就是说，35个特征只能在我们的设置中增加），或者由于所请求的变化的不可行性。

基于学习的方法不安全的根本问题在于特征的设计。机器学习在各种信息系统中越来越受欢迎 - 这远远超出了安全 - 主要是因为它能够预测，或多或少的成功，从*副作用的原因*。使机器成为这种概括性的能力

**（补全预防措施，加入实验数据）**

（加入 重点特征的分析，详见PE论文 Table 1：Descriptions of Some Important Features）

（加入 低延迟：维持在ms级别？，高吞吐率：整个系统每天能处理多少恶意样本）

（考虑加入：预测恶意/好文件 的时间以及比较）

**（考虑加入：APK Paper Figure 4. Run-time performance）**

**讨论：**

1 动态分析的检测：

我们使用了一些具有恶意链接调用的PDF进行动态检测，我们可以看到正常的恶意文件是很少有URL调用的，恶意的PDF文件一般会通过使用暗链连接外部数据，从而下载恶意软件，对客户端进行有目的性的攻击。在对这部分文件进行检测时，因为文件往往是有针对性的，所以对一些特定的攻击与攻击环境很难模拟到，以至于在沙箱里面运行的时候，检测不到恶意行为，于此同时以工业性的角度来看，动态检测不仅会占用大量资源，内存，这样一来就与沙箱检测的环境要求还是很高的，并且沙箱检测的时间远远的高于静态检测的时间，不利于在线检测的一个实时返回的一个结果。对于静态检测说，主要针对静态特征的提取，以及文件格式和内容的静态代码分析，

## 总结

## 参考文献

1. Davide Balzarotti, Marco Cova, Christoph Karlberger, Christopher Kruegel, Engin Kirda, and Giovann Vigna. Efficient Detection of Split Personalities in Malware. In Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February–March 2010
2. Curtis Carmony, Mu Zhang, Xunchao Hu, Abhishek Vasisht Bhaskar, and Heng Yin. Extract Me If You Can: Abusing PDF Parsers in Malware Detectors. In Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February 2016
3. Nedim ˇ Srndic and Pavel Laskov. Detection of Malicious Pdf Files Based on Hierarchical Document Structure. In 20th Network and Distributed System Security Symposium (NDSS), 2013
4. Nedim Srndic and Pavel Laskov. Practical Evasion of a Learning- Based Classifier: A Case Study. In Proceedings of the 35th IEEE Symposium on Security and Privacy (Oakland), San Jose, CA, May 2014
5. Nedim ˇ Srndic and Pavel Laskov. Mimicus: A Library for Adversarial Classifier Evasion. <https://github.com/srndic/mimicus>.
6. Nedim Šrndic and Pavel Laskov . Hidost: a static machine-learning-based detector of malicious files, Šrndi′c and Laskov EURASIP Journal on Information Security (2016) 2016
7. Weilin Xu, Yanjun Qi, and David Evans. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. In Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February 2016.
8. VirusTotal. Free Online Virus, Malware and URL Scanner.https://www.virustotal.com/.
9. Stephan Chenette. Malicious Documents Archive for Signature Testing and Research - Contagio Malware Dump. http://contagiodump.blogspot.de/2010/08/ malicious-documents-archive-for.html.
10. D. Maiorca, D. Ariu, I. Corona, and G. Giacinto, “A Structural and Content-Based Approach for a Precise and Robust Detection of Malicious PDF Files”, in Proceedings of the 1st International Conference on Information Systems Security and Privacy. ScitePress Digital Library,2015, pp. 27-36.
11. Charles Smutz and Angelos Stavrou. Malicious PDF Detection Using Metadata and Structural Features. Technical report, 2012.

Maiorca等人。也研究了对PDFrate和其他PDF文档分类器的逃逸[26]，[27]。他们提出 反向模仿技术。为了使内容看似良性（如Mimicus所做的那样），他们将恶意内容嵌入到良性PDF中，而不是将内容添加到恶意文档中，而是尽可能少地修改。反向模仿攻击实施针对PDFrate的独立逃避方法, Maiorca等人提出了三种不同的逃避方案。在EXEembed方案中，恶意可执行文件被植入到现有的良性PDF文档中。打开文档时会执行恶意软件。这些文档利用CVE-2010-1240。在PDFembed方案中，恶意PDF被嵌入到良性PDF中。这些嵌入式文档在文档打开时自动呈现。为了评估，Maiorca等人 将利用CVE-2009-0927的文档嵌入到现有的良性PDF文档中。最后，在JSinject场景中，恶意JavaScript（与PDFembed嵌入式文档中使用的相同）直接注入根良性文档。

## 附录：

|  |  |
| --- | --- |
| pdfid0\_len | 0.086690379 |
| pdfid1\_mismatch | 0.084174646 |
| pdfid1\_len | 0.05768113 |
| pdfid0\_num | 0.054554554 |
| pdfid1\_num | 0.046312153 |
| producer\_mismatch | 0.043095205 |
| pdfid0\_mismatch | 0.041989514 |
| size | 0.036038988 |
| len\_obj\_min | 0.028280255 |
| createdate\_ts | 0.025696944 |
| len\_stream\_avg | 0.021415648 |
| producer\_uc | 0.021115434 |
| len\_obj\_avg | 0.019194021 |
| creator\_dot | 0.019165413 |
| count\_javascript | 0.018683933 |
| createdate\_tz | 0.018423179 |
| count\_font | 0.018205831 |
| count\_obj | 0.017008854 |
| pdfid1\_uc | 0.01506473 |
| pdfid\_mismatch | 0.013957534 |
| producer\_dot | 0.013802704 |
| len\_stream\_max | 0.013243159 |
| count\_endobj | 0.01301985 |
| author\_mismatch | 0.012865107 |
| count\_js | 0.01283716 |
| count\_stream | 0.012261984 |
| len\_obj\_max | 0.01173477 |
| count\_endstream | 0.011612836 |
| pdfid0\_uc | 0.011072068 |
| creator\_oth | 0.010152634 |
| count\_acroform | 0.008933302 |
| version | 0.008247037 |
| creator\_uc | 0.008067628 |
| count\_startxref | 0.007172574 |
| moddate\_version\_ratio | 0.006797005 |
| moddate\_ts | 0.00576873 |
| delta\_tz | 0.005767478 |
| producer\_oth | 0.00552752 |
| count\_page | 0.005345291 |
| image\_totalpx | 0.005311085 |
| creator\_len | 0.005305406 |
| moddate\_tz | 0.005187586 |
| count\_eof | 0.005069558 |
| creator\_num | 0.005004477 |
| producer\_num | 0.004567713 |
| count\_trailer | 0.004553001 |
| createdate\_version\_ratio | 0.004431596 |
| count\_box\_other | 0.004428159 |
| producer\_lc | 0.004385414 |
| title\_dot | 0.003985462 |
| creator\_lc | 0.003918669 |
| count\_stream\_diff | 0.003675752 |
| producer\_len | 0.003499005 |
| pdfid1\_lc | 0.003496048 |
| count\_image\_large | 0.003015509 |
| title\_len | 0.00301476 |
| creator\_mismatch | 0.00299044 |
| count\_action | 0.002914612 |
| len\_stream\_min | 0.002904229 |
| count\_image\_total | 0.002903827 |
| box\_other\_only | 0.002436496 |
| author\_len | 0.002286143 |
| delta\_ts | 0.002171871 |
| title\_oth | 0.00213326 |
| title\_lc | 0.00195717 |
| author\_oth | 0.001922557 |
| pdfid0\_lc | 0.001667383 |
| box\_nonother\_types | 0.001663975 |
| count\_xref | 0.001624368 |
| keywords\_len | 0.001472989 |
| moddate\_mismatch | 0.001469648 |
| title\_uc | 0.001187556 |
| keywords\_lc | 0.001184722 |
| pos\_acroform\_min | 0.001167811 |
| pos\_acroform\_max | 0.001147658 |
| author\_uc | 0.00110271 |
| count\_box\_letter | 0.001076087 |
| count\_image\_med | 0.001051931 |
| pos\_acroform\_avg | 0.001022001 |
| title\_num | 0.001009777 |
| subject\_uc | 0.000943209 |
| author\_lc | 0.000909401 |
| keywords\_mismatch | 0.000887583 |
| keywords\_uc | 0.00085437 |
| subject\_oth | 0.000820395 |
| keywords\_num | 0.000782946 |
| author\_num | 0.000728195 |
| subject\_lc | 0.000722817 |
| subject\_len | 0.000707868 |
| subject\_mismatch | 0.000556082 |
| image\_mismatch | 0.000459965 |
| count\_image\_small | 0.000452753 |
| subject\_num | 0.000378177 |
| pos\_image\_min | 0.000368489 |
| title\_mismatch | 0.00036401 |
| ratio\_imagepx\_size | 0.000351227 |
| pdfid0\_oth | 0.000339025 |
| keywords\_oth | 0.000331225 |
| pos\_image\_avg | 0.000283184 |
| pos\_image\_max | 0.000265902 |
| count\_javascript\_obs | 0.00025581 |
| pdfid1\_oth | 0.000246812 |
| createdate\_mismatch | 0.000240992 |
| count\_page\_obs | 0.000229257 |
| company\_mismatch | 0.000213937 |
| count\_objstm | 0.000178874 |
| author\_dot | 0.000155896 |
| count\_js\_obs | 0.000139022 |
| subject\_dot | 8.93E-05 |
| count\_box\_a4 | 8.58E-05 |
| count\_font\_obs | 6.84E-05 |
| count\_image\_xsmall | 5.30E-05 |
| count\_acroform\_obs | 4.52E-05 |
| count\_action\_obs | 3.93E-05 |
| pos\_page\_max | 3.53E-05 |
| pos\_page\_avg | 2.10E-05 |
| keywords\_dot | 2.09E-05 |
| pos\_page\_min | 1.84E-05 |
| pos\_box\_avg | 1.55E-05 |
| ratio\_size\_stream | 1.09E-05 |
| ratio\_size\_page | 8.86E-06 |
| pos\_box\_max | 8.23E-06 |
| count\_image\_xlarge | 5.37E-06 |
| count\_box\_legal | 5.23E-06 |
| pos\_box\_min | 5.13E-06 |
| ratio\_size\_obj | 9.81E-07 |
| count\_box\_overlap | 3.31E-07 |
| count\_objstm\_obs | 0 |
| createdate\_dot | 0 |
| moddate\_dot | 0 |
| pdfid0\_dot | 0 |
| pdfid1\_dot | 0 |
| pos\_eof\_avg | 0 |
| pos\_eof\_max | 0 |
| pos\_eof\_min | 0 |