# 1 背景

去年听学长说 2018 秋招的 AI 岗位竞争挺激烈的，今年又听学长说 2019 秋招的 AI 岗位很激烈，原因无非是做 AI 方向的同学越来越多，而且很多都是跨专业的。大环境越来越激烈，但是一些公司表示还是招不到满意的人。那么未来几年内，哪个方向的机器学习人才最紧缺呢？我觉得有两个方向，第一个方向是机器学习+垂直领域，第二个方向是机器学习落地。早几年 AI 行业的招聘是走粗放模式的，只要你是调包侠调参怪，学历不错，都有好的 offer，如今 AI 领域的岗位正在朝精细化发展，特定领域的机器学习专家最紧缺，这就是机器学习应用于垂直领域，所有的跨领域应用最终都是为了机器学习落地产生商业价值，这就需要全栈式机器学习人才，从需求、模型、算法、高效工程实现、部署和优化一把梭做好的机器学习人才。

本篇文章从机器学习在垂直领域的应用出发，具体阐述机器学习在网络空间安全领域的应用。

# 2 机器学习在网络空间安全垂直领域的应用

我国于 2015 年正式批准设立“网络空间安全”国家一级学科，目前网络空间安全的研究主要涉及五个研究方向，即网络空间安全基础、密码学及应用、系统安全、网络安全、应用安全。

目前机器学习在网络空间安全基础、密码学及应用两个方向的研究较少涉及，而在系统安全、网络安全、应用安全三个方向中有大量的研究。其中，系统安全以芯片、系统硬件物理环境及系统软件为演技对象，网络安全主要以网络基础设施、网络安全监测为研究重点，应用层面则关注应用软件安全、社会网络安全。

个人比较关注网络安全和应用安全，而第三届阿里云安全算法挑战赛的主题是恶意软件分类，属于应用安全，所以我尝试以本次比赛为例分析机器学习在安全垂直领域的应用。

# 3 第三届阿里云安全算法挑战赛

第三届阿里云安全算法挑战赛基本介绍：使用来自 windows 可执行程序经过沙箱程序模拟运行后得到的 API 指令序列数据集，区分五类恶意软件。竞赛背景、数据说明等详细介绍参考天池第三届阿里云安全算法挑战赛。

我注意到此次比赛组队规则中有一句：鼓励算法背景选手和安全背景选手跨界组合。这或许正验证着机器学习和垂直领域跨领域结合的深度问题。仅有算法背景的同学和有跨领域背景的同学同时去做垂直领域，或许结合的深度有所不同，而产生意想不到的结果。所以我分别从算法工程师的算法角度和安全算法工程师的安全算法两个角度去分析此次比赛。

## 3.1 第一层面：算法

仅从算法工程师（无安全领域背景）的层面分析待解决安全问题，浅层次结合机器学习和垂直领域

### 3.1.1 探索性数据分析

在建模之前，我们都会对原始数据进行一些可视化探索，以便更快地熟悉数据，更有效进行之后的特征工程和建模

打开训练集，取前 5 行观测

```
data_df.head()
```

| | file_id | label | api | tid | return_value | index |
|---|---|---|---|---|---|---|
| **0** | 0 | 0 | GetSystemTimeAsFileTime | 2644 | 0 | 0 |
| **1** | 0 | 0 | NtAllocateVirtualMemory | 2644 | 0 | 1 |
| **2** | 0 | 0 | NtFreeVirtualMemory | 2644 | 0 | 2 |
| **3** | 0 | 0 | NtAllocateVirtualMemory | 2644 | 0 | 3 |
| **4** | 0 | 0 | NtAllocateVirtualMemory | 2644 | 0 | 4 |

我们可以看到给定的数据中包含数值型和文本型的特征

看一下训练集有多少数据

```
data_df.shape
```

```
(409631049, 6)
```

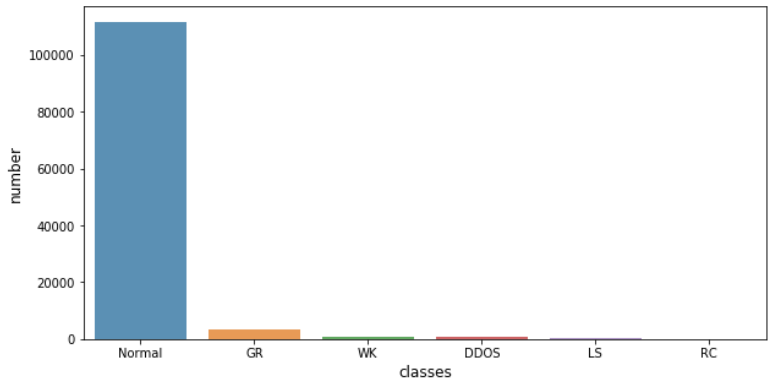
训练集有 409631049 个样例

我们先看看目标变量 label

```
label_class
```

```
[0]  111545
[5]    3397
[2]     744
[3]     598
[1]     287
[4]      53
Name: label, dtype: int64
```

绘成柱状图




感染型病毒文件在五类恶意软件中数量最多，有 3397 个，其次是挖矿程序文件，有 744 个，DDOS 木马文件有 598 个，勒索病毒文件有 287 个，蠕虫病毒文件最少，只有 53 个。

再看看 api 的数量分布

```
data_df['api'].value_counts()
```

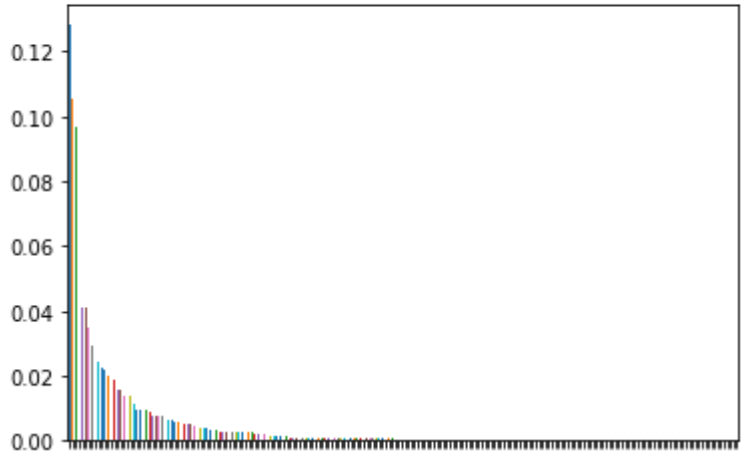
```
RegQueryValueExW              47244792
LdrGetProcedureAddress        28991443
NtClose                       28662647
RegOpenKeyExW                 20731738
NtReadFile                    20294917
CryptDecodeObjectEx           19773656
GetSystemMetrics              18634068
RegCloseKey                   15709773
NtAllocateVirtualMemory       13012425
NtQueryValueKey               10512378
NtWriteFile                    8435148
NtDelayExecution               8147888
GetCursorPos                   7765612
NtQueryKey                     6785908
LdrGetDllHandle                5621932
NtOpenKeyEx                    5615435
NtCreateFile                   5424751
Thread32Next                   5261448
NtOpenKey                      5097316
GetSystemTimeAsFileTime        4956651
NtFreeVirtualMemory            4929259
GetKeyState                    4918436
ReadProcessMemory              4469477
NtQueryDirectoryFile           4465276
LdrLoadDll                     4364239
SetFilePointer                 4192066
LoadResource                   3858739
GetFileAttributesW             3146108
SetErrorMode                   3110380
RegEnumKeyExW                  3071760
```

可以看到排名靠前的 api 主要功能是注册表读写、文件读写、获取系统信息

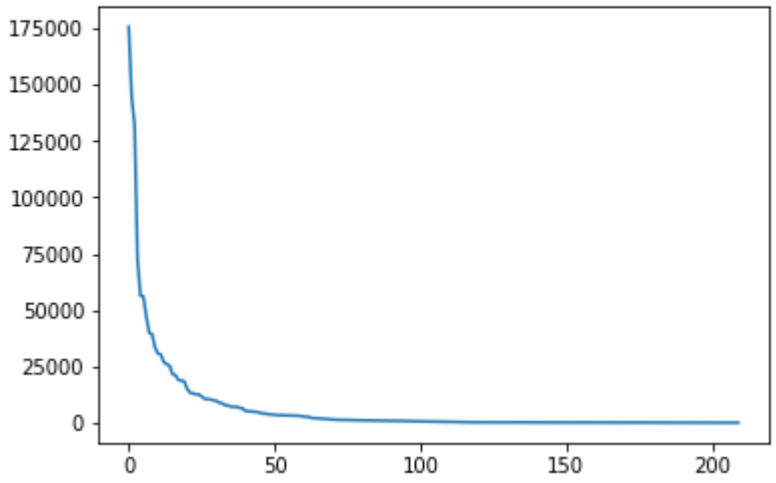
再看看不同 label 的 api 数量分布，

看第一类勒索病毒 api 数量分布

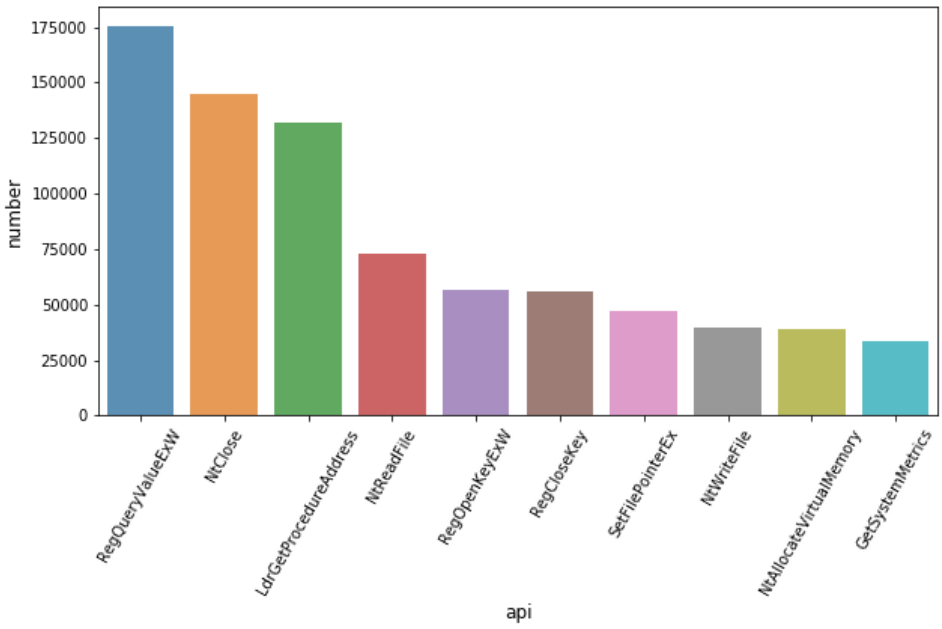
```
(api/api.sum()).plot.bar()
plt.show()
```




```
import matplotlib.pyplot as plt
plt.plot(api.values)
plt.show()
```




看一下 top10 api




### 3.1.2 赛题理解

预分析完数据后，再结合整体比赛介绍，我们来还原一下比赛的数据集是如何产生的：假设某个文件有以下的 api 调用顺序，采集精度是 10ms，这里忽略 api 的 return value

| index | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| timestamp(ms) | | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| tid | 1001 | a | a,a | | | a | a | a | | | c | a |
| | 1002 | | b | c | | | d | | | | | b |
| | 1003 | f | | | | f | | | | | f | |

那么将生成如下数据文件

```
api     tid     index
a       1001    0
a       1001    1
a       1001    1
a       1001    3
a       1001    4
a       1001    5
c       1001    6
a       1002    7
b       1002    1
c       1002    2
d       1002    4
b       1002    7
f       1003    0
f       1003    3
f       1003    6
```

### 3.1.3 抽象问题

每个文件有多个 api 调用，api 之间可能存在一定的序列关系，同时 api 可以看成是一个个单词，这样文件的 api 调用就是一个个文本，所以要解决的分类问题可以看成是 NLP 领域中含单词顺序的文本分类问题。

### 3.1.4 解决问题

解决问题的方式大概有三种：

- 就用 NLP 领域的文本词带模型来建模；
- 用统计机器学习中的传统统计特征来建模分析；
- 从深度学习领域处理时间序列的序列建模角度进行训练模型

现在来解决问题：

1） 仅使用 NLP 领域的文本词带模型训练模型

特征提取和算法分别使用 n-gram tfidf 和 xgboost，其中 n-gram 可分为 unigram、bigram ……9-gram 、10-gram。做了四组实验，如表 1 所示：

表 1（注：表中 mlogloss 是原始的 logloss，loss 是官方定义的 loss）

| NLP | unigram tfidf | Bigram tfidf | Trigram tfidf | Uni-bi-trigram tfidf |
|---|---|---|---|---|
| feature numbers | 308 | 30984 | 50000 | 81292 |
| Train mlogloss | 0.00634116 | 0.00301971 | 0.00332867 | 0.00332701 |
| Train loss | 0.0123793 | 0.00600566 | 0.00662049 | 0.006617 |
| valid mlogloss | 0.0359148 | 0.0261557 | 0.0276649 | 0.0269315 |
| valid loss | 0.0654428 | 0.0479153 | 0.0508107 | 0.0493288 |

可以看到单纯的文本向量已经拥有不错的效果，bigram 效果最好，unigram 到 bigram，loss 有较明显的下降，优化比较明显，再往下到 trigram 优化效果反而不好，结合三种 gram，较 trigram 效果略有提升。分析下原因：可能是由于文件的 api 之间存在一定的时间序列关系，所以当使用含有一定的序列建模能力的 bigram 时，效果会得到提升，但是可能 api 之间的序列关系太长了，可能分布几百到几十万不等，使用 n-gram tfidf 很难对长序列建模，序列关系不确定可能导致 trigram 效果不好。

2) 仅使用统计机器学习领域的统计特征训练模型

学习了排行榜第 55 名、第 7 名和第 1 名的队伍的统计学特征提取方式，发现提取的统计特征越全面、越多，模型效果一般越好，这里比较下三个队伍在统计学领域对本次比赛中安全数据的处理方式。

55 th 的队伍提取了一些全局特征和局部特征，如表 2；7 th 的队伍提取的统计特征为全局特征和 api 特征，全局特征相比 55th 并没有大的改变，关键在于新增的 api 特征：每个文件中的 api 在总 api 中出现的次数和比率。总 api 去重后有 300 个左右，按照这种特征处理方式提取特征，增加了 600 多个 api 特征；1 st 的队伍提取的统计特征非常全面，主要分为四大类：全局特征、局部组合特征、高阶局部特征、2-gram 局部特征展开。和之前两只队伍比，全局特征和局部组合特征基本不变（全局特征相比 55th 增加了 quantile 分位数），主要加入了高阶局部特征和 2-gram 局部特征，如表 3。

表 2：全局特征和局部特征

| 全局特征 | 局部特征 |
|---|---|
| file_id(api):count、nunique | file_id+api(tid):min 、 max 、 mean 、 median、std、mad、skew、kurt |
| file_id(tid):count、nunique、max、min、median、mean、std | file_id+api(returnvalue):min、max、mean、median、std、mad、skew、kurt |
| file_id(returnvalue):count、nunique、max、min、median、mean、std | file_id+tid(api):min、max、mean、median、std、mad、skew、kurt |
| file_id(index):count | file_id+tid(returnvalue):min、mean、median、std、mad、skew、kurt |

表 3:高阶局部特征和 2-gram 局部特征

| 高阶局部特征 | 2-gram 局部组合特征（展开形式） |
|---|---|
| file_id+api+tid(returnvalue):nunique、max、min、median、std | file_id+api_2 |
| file_id+api+tid(index):nunique、max、min、median、std | file_id+api_2(tid):count、nunique |
| file_id+tid+api(returnvalue):nunique、max、min、median、std | file_id+api_2(returnvalue):nunique,max,min,median,std |

| | |
|---|---|
| file_id+tid+api(index):nunique、max、min、median、std | file_id+api_2(index):nunique、max、min、median、std |

测试了三组实验，如下表：

| Statistics | 55th | 7th | 1st |
|---|---|---|---|
| feature numbers | 60+ | 635 | 3700+ |
| train mlogloss | 0.00653823 | 0.00311932 | 0.00237912 |
| train loss | 0.0130206 | 0.00623123 | 0.00474234 |
| valid mlogloss | 0.0538816 | 0.0336371 | 0.0194609 |
| valid loss | 0.0977158 | 0.0614191 | 0.0357663 |

分析实验结果：可以看到统计特征越全面越多效果一般越好，全面的统计特征效果好于 tfidf 效果。存在的问题是较难全面地提取统计特征，包括但不限于全局特征、局部特征、高级局部特征等等，并且有些特征可能会过拟合，操作起来有点麻烦

3) 结合 NLP 特征和统计学特征训练单模型

合并第 55th 队伍的统计学特征和 NLP 下的特征训练单模型

| Statistics+NLP | feature numbers | Train loss | Valid loss |
|---|---|---|---|
| 55th+unigram+tfidf | 368+ | 0.00711616 | 0.0662567 |
| 55th+bigram+tfidf | 31044+ | 0.00544134 | 0.0505155 |
| 55th+trigram+tfidf | 50060+ | 0.00540732 | 0.0515178 |
| 55th+uni+bi+trigram+tfidf | 81352+ | 0.00540852 | 0.0511642 |

对比表 1，观察 train loss 下降了，valid loss 不但没有下降，反而提升了一点，原因应该是 tfidf 的文本向量的效果已经不错，加入传统简单统计特征会过拟合（这里有疑惑，7 th 队伍就没有过拟合），但是由于特征的可解释性问题，具体无法解释。目前只用 lgb 实验了合并两个领域的特征向量训练单模型，会过拟合。未实验模型融合的方法，即分别使用两个领域的特征训练两个单模型再模型融合，猜想效果会好些，应该不会过拟合。

合并 7th 队伍的统计学特征和 NLP 下的特征训练单模型

| Statistics+NLP | feature numbers | Train loss | Valid loss |
|---|---|---|---|
| 7th+unigram+tfidf | 943 | 0.00645729 | 0.0616332 |
| 7th+bigram+tfidf | 31619 | 0.00480952 | 0.0480674 |
| 7th+trigram+tfidf | 35885 | 0.00549029 | 0.0495199 |

对比上表，观察加入 55th 的统计特征，valid loss 有一定的回升，说明 7th 加入的 api 特征起到了作用，加入好的统计特征对效果不错的 tfidf 有一定的促进作用，加入差的统计特征反而效果不好。到这里为止的模型都无法对 api 进行长序列建模，

但是还是可以通过 api 的数量和占比等传统统计特征来提升检测效果，现在欠缺的就是长序列建模。

没有测试 1st 的统计学特征和 NLP 下特征合并训练单模型，猜测模型效果较以上会得到明显提升。个人感觉使用 tfidf 复杂度较低，同时模型效果还不错，可以作为 baseline，然后再尝试统计学统计特征训练模型对比效果。

4) 从深度学习领域处理时间序列的序列建模角度进行训练模型

深度学习这块还没有细加研究，有的队伍使用的是对 api sequence 进行 One-Hot Encoding ,变成 2D Matrix，然后使用 CNN 去训练模型，效果一般，具体可参考 https://github.com/beader/tianchi-3rd_security/

因为 api 序列长度分布几百到几十万不等，而且存在明显的时序性，传统 RNN 网络很难对长序列建模，所以第一名的队伍采用的是 textcnn（一种利用 cnn 对文本分类的算法），textcnn 通过扩大 kernel_size 可以一定上弥补该问题，但是依旧很难进行较长的序列建模。

### 3.1.5 1st 的解决方案

表：三只队伍的对比

| 覆盖点 | 55 th | 7 th | 1 st |
|---|---|---|---|
| NLP | × | √ | √ |
| 统计学 | √ | √ | √ |
| 深度学习 | × | × | √ |
| 模型融合 | × | × | √ |
| 安全知识 | × | × | 未知 |

其中安全知识这块，1 st 在 ppt 中提到了两点安全背景：利用数据流分析 API 之间的数据依赖关系，生成特征向量，再运用机器学习的分类算法，进行恶意软件的检测和分类；基于关联规则分析 Aprori 算法，根据敏感 API 和恶意软件类型之间的数据关系的调用来识别。但是我没有具体找到利用安全背景提取特征的痕迹

可以看到 1st 的队伍从多个领域对本次比赛做了细致的处理，涵盖了几乎所有可能的处理方式，每种方式处理的效果都很优秀。

仅从算法角度我先得到了几点推论：

1、NLP 可以用来做 baseline，值得偏重

2、要从多维度多领域看待问题解决问题，全面发展

3、没有太多的垂直领域背景，可以达到较好的效果

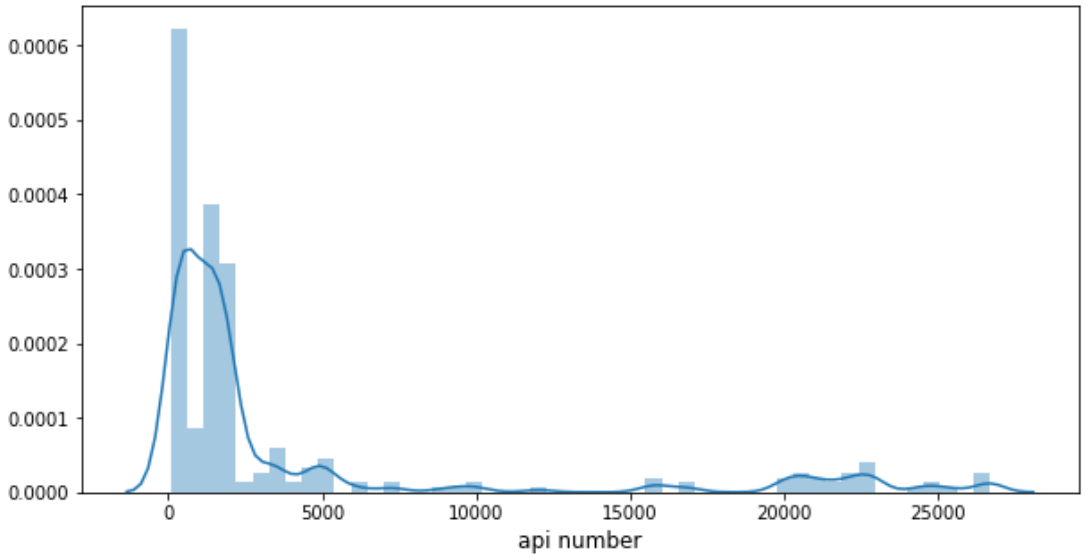
## 3.2 第二层面：安全算法

从安全算法工程师的角度，尝试利用安全领域知识和算法知识去分析和解决问题，深层次结合机器学习和垂直领域，这种结合我认为可分为三个阶段。

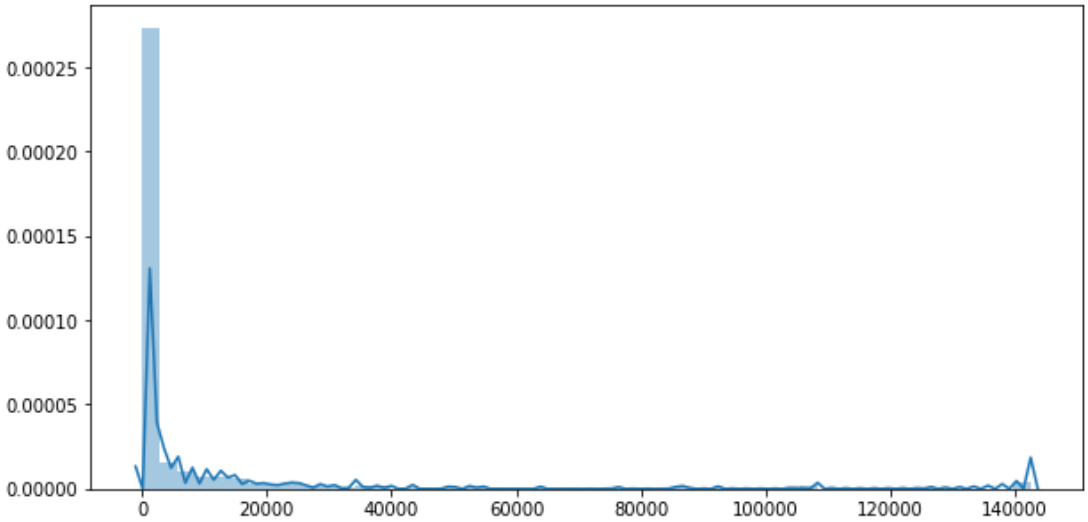
### 3.2.1 数据级可解释

从安全角度分析安全数据，首先要做到的是联系安全数据对应上安全领域知识，作为安全研究的辅助手段，促进对安全理解和研究。

先观察一下第一类勒索病毒 api 数量分布直方图。

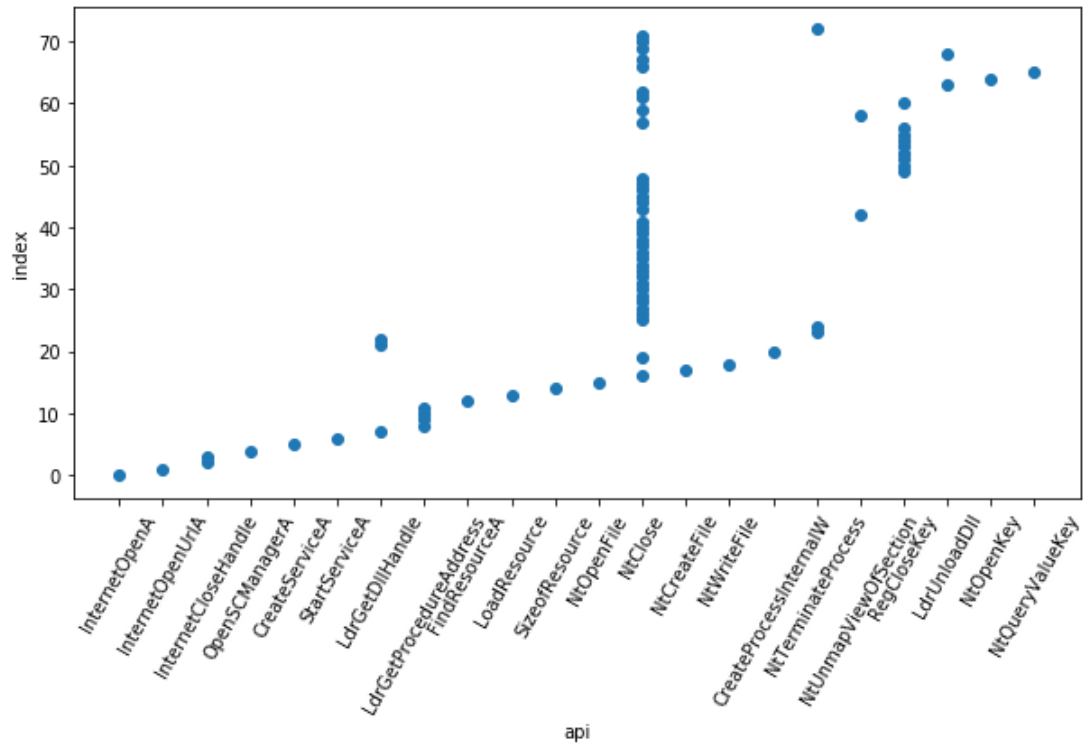


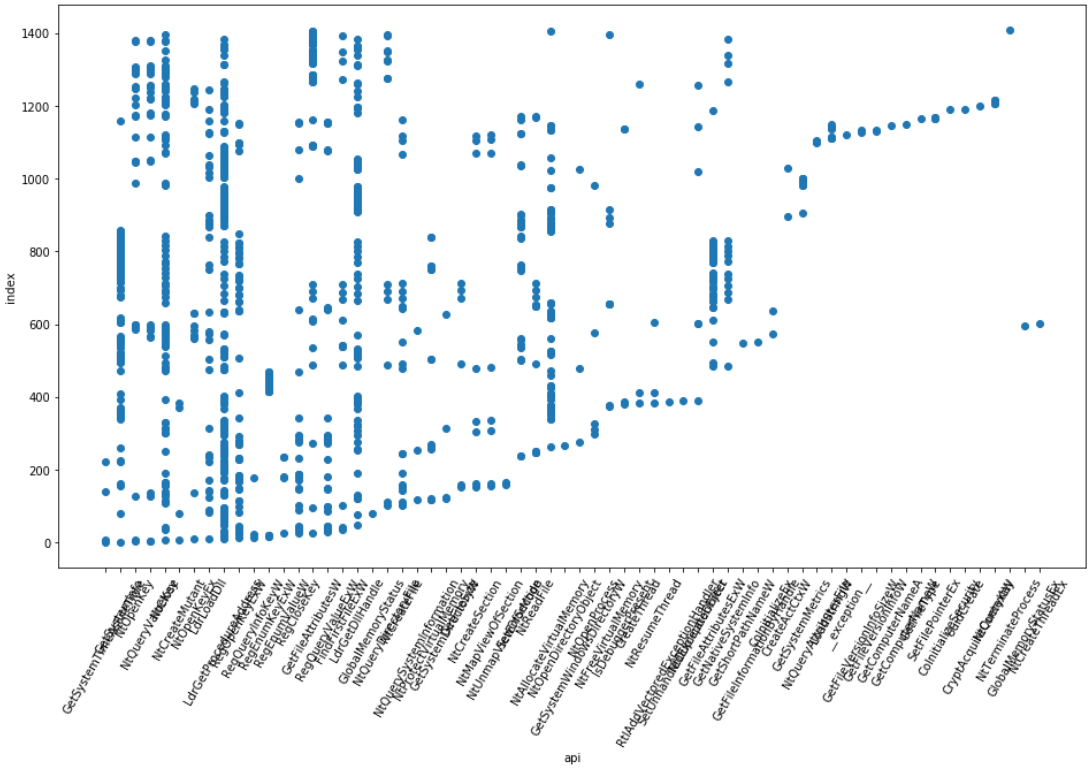

再对比一下挖矿恶意软件




依次绘出其他类型恶意软件的直方图，对比一下，发现勒索病毒在五类恶意软件中调用的 api 数量大多较少，蠕虫和感染性病毒 api 调用数量较勒索病毒多，挖矿程序和 DDOS 木马调用 api 数量较多。这符合我这系统安全的小白对五类恶意软件的了解

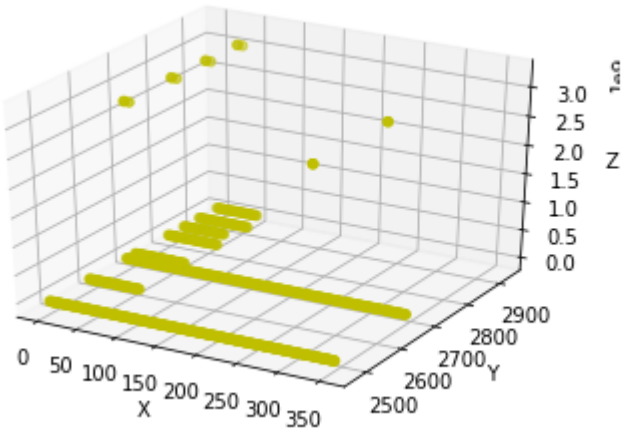
挑选五类恶意软件中 api 长度 1%分位，25%分位，50%分位，99%分位的文件对比分析，绘出二维散点图，以勒索软件为例。



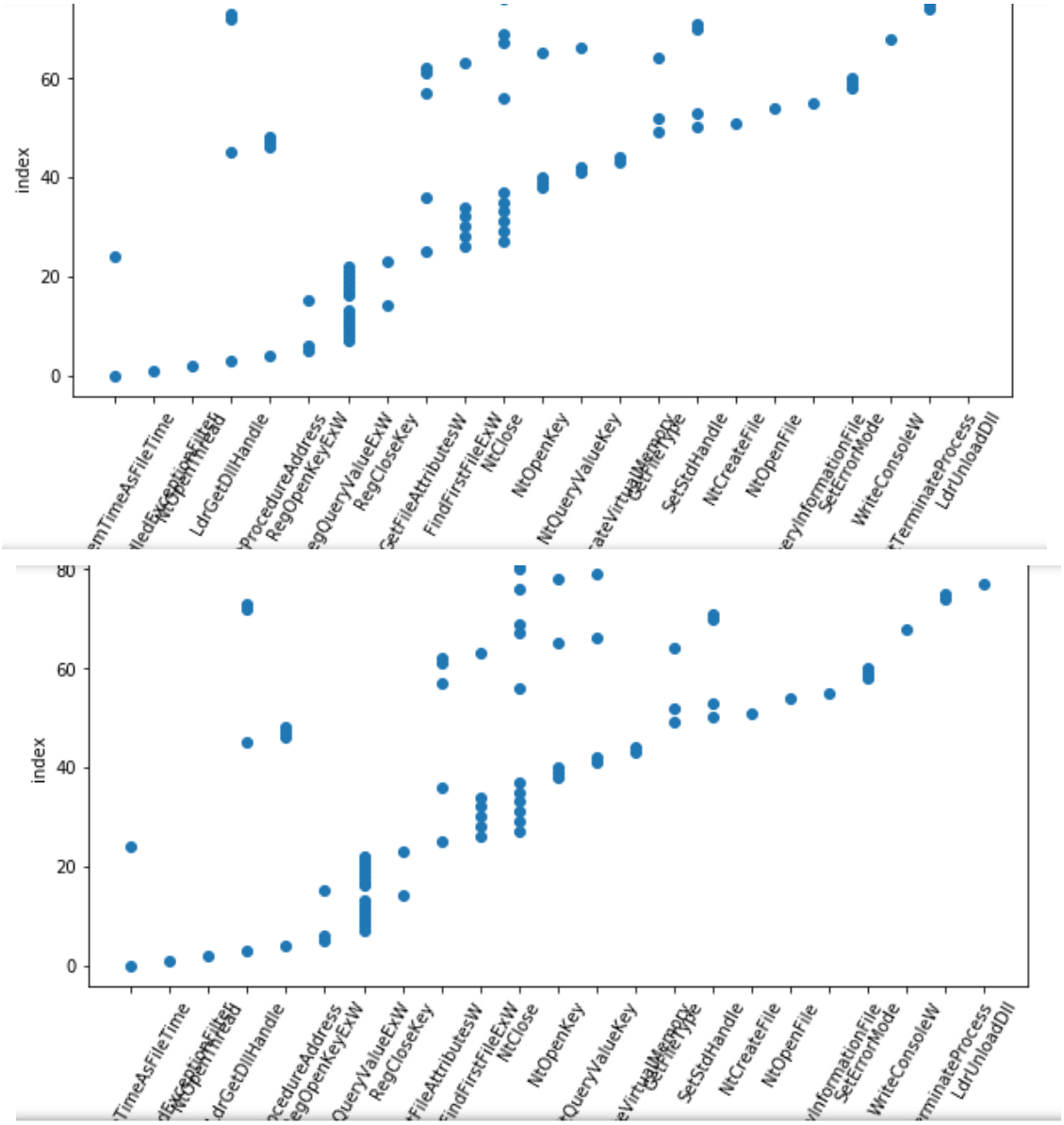

结合散点图和勒索软件的 top10 api，体现出了勒索软件的一些行为习性，即勒索病毒会进行大量的注册表读写和文件读写操作。

挑选五类恶意软件中 api 长度 1%分位，25%分位，50%分位，99%分位的文件对比分析，绘出三维散点图，以挖矿程序为例




可以看到挖矿程序的三维图非常有规律，对应到安全领域知识是挖矿程序的多线程特征，api 调用有一定的时间序列关系。

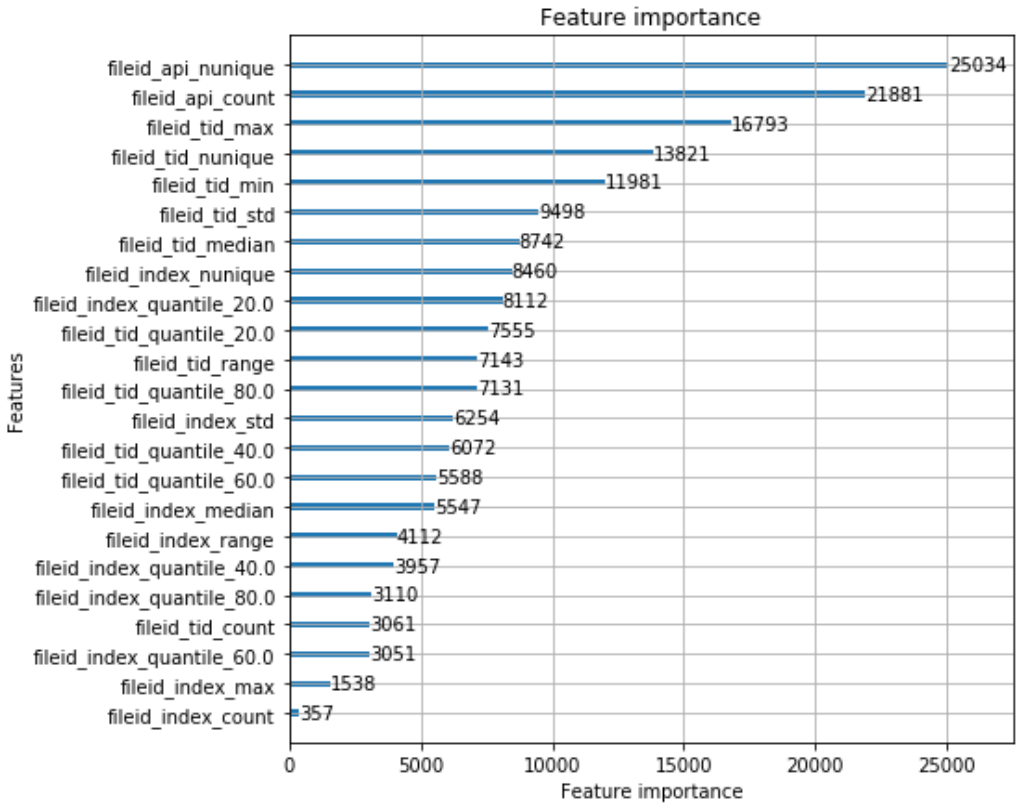
上面都是由数据来对应安全领域知识，现在我们从安全领域知识出发去找到对应的数据，以蠕虫病毒为例。我们知道蠕虫的特点是复制自身，感染其他机器，具有传播性，现在我们寻找这种数据




可以看到这个文件中出现了多个同样的 api 序列同时在读写文件，这可能就是我们要寻找的蠕虫自我复制的数据。蠕虫的感染性体现在 top api 中的 Thread32Next 函数和 LdrLoadDll 函数。但是蠕虫的传播性我没有找到证明的数据。

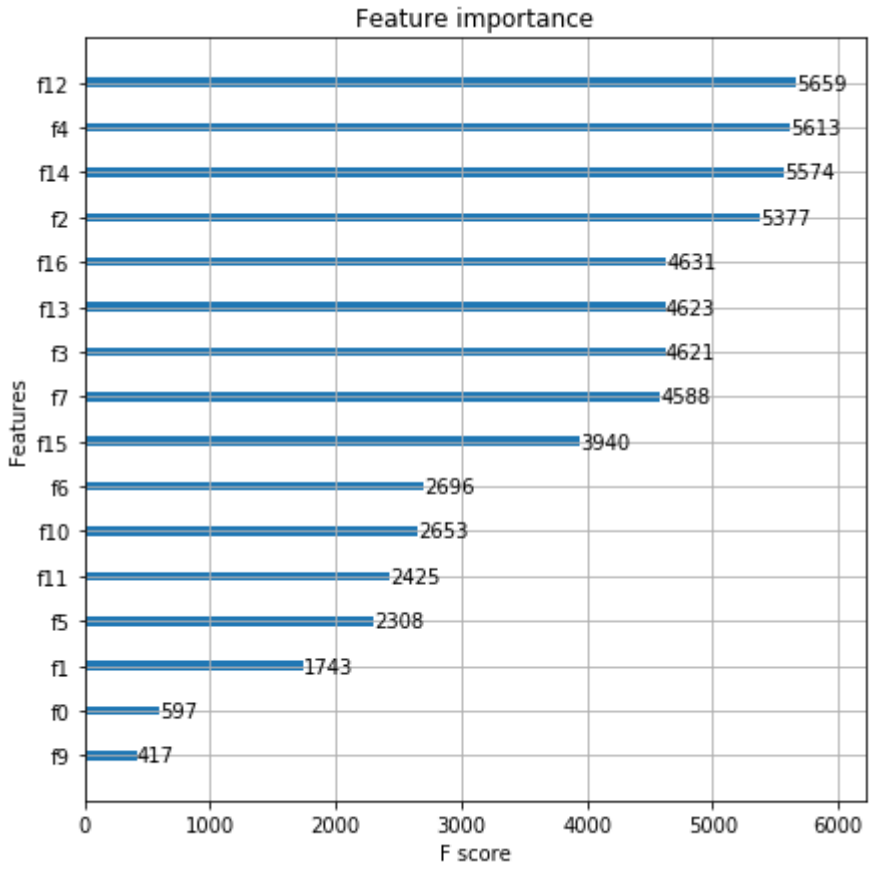
### 3.2.2 特征级可解释

通过安全领域知识（例如此次恶意软件检测比赛中如 1st ppt 中提到的两点安全背景）反哺，深度分析数据性质，重构特征，达到特征可解释，提升模型效果，解决安全问题或可能促进安全研究。




先可视化全局特征看一下各个特征的重要性，可以发现 top2 的特征都是和 api 相关的，这正印证着安全背景知识中恶意软件重要的 api 行为特征，并且后续加入 api 相关的特征可以较好的提升模型效果，比如 7th 队伍增加的 300 多个 api 统计特征。本次数据中属性列只有五个，不需要太多安全背景就能分析出关键点在 api 属性，如果数据集属性较多，就需要基于安全背景知识去快速定位问题关键所在，然后重点做关键属性的特征工程。

比如 Kaggle 上的 Rental Listing Inquiries 比赛，这是一个租房数据集，数据集里面的 15 个属性值都有一定的作用，但是无法得知哪个或哪些属性值最关键，如果直接做简单特征重要性可视化




虽然 top2 特征是 f12 和 f4，但是这并不是问题的关键点，在这两个特征上再做特征工程也提升不了太多效果，而最关键的是 f14 特征（manager_id(经理 id)）排第三。如果关键特征或属性在 top 特征中，还可以凭借算法知识挨个实验，分析出问题关键所在，但如果关键属性不在 top 特征，可能就需要相关的背景知识来分析问题了，例如此处如果有租房界工作经历的人学了机器学习，就可能直接凭借背景经验得出 manager_id 是重要特征，快速在 manager_id 特征之上做各种复合特征。

存在的问题是我举的例子可能属于个例，机器学习中特征的可解释性还是很弱，深度学习的特征可解释性压根没得解释。

## 3.3 算法 vs 安全算法

本文中所举例的第三届阿里云安全算法挑战赛，我的最终的实验结果表明：安全算法和算法的效果并没有太大的区分性，这没有达到我预期的效果。我的预期效果是：在安全领域，安全算法效果胜于算法效果。因为安全算法专家可以根据特征级解释有针对性的去做特征工程，比如根据领域知识知道 api 序列特征非常重要，重点去做 api 相关的特征，而不是纯粹从统计学等领域去尝试各个特征，无法直接定位问题关键点。没有达到预期效果的原因可能是目前水平有限，分析解决问题做的不是很到位，无法给算法加持太多的安全领域知识，以形成鲜明的对比，或是处理此次比赛安全数据集所需要的领域知识的门槛不是很高，大家都知道 api 序列很关键，感觉此次比赛中安全领域知识发挥的作用不是很大。

排行榜前几名的队伍好像都是职业做算法的，说明算法水平很强可以弥补安全背景的不足，不过也可能是参赛的复合背景队伍较少，因为大家的算法水平不一，所以不太好比较。

我认为从单个安全问题和模型效果角度来说，算法工程师只需要针对单个待解决的安全问题临时补习一下安全基础知识，就可以较好的处理安全数据，所以算法工程师的可选择性有很多，较容易跨入安全等垂直领域，但是不能浅尝辄止，需要深入到一个领域中去；如果从模型效果、模型落地和安全研究的三重角度来说，算法工程师只能解决单个的安全问题，很难做到对安全系统有全面的深度了解，还是需要安全出身的算法专家，安全算法专家的安全知识面和算法知识面都较广，能够全面把握安全问题，根据安全场景选择合适的算法并推动该算法在 HIDS、NIDS、RASP、WAF 等安全系统里的落地实现。所以我们要做某个垂直领域最懂机器学习的人。

### 3.4 其他

- 关于机器内存问题：小内存机器处理大数据集，可采用 python 的生成器和机器学习算法的 fit_generator 函数或是 Linux 的 awk 预处理大数据集
- 关于 lightGBM：本文使用的都是 lgb，效果对比 xgb，train logloss 差别不大，valid logloss 相差挺大，而且还是 xgb 效果好于 lgb（一般 lgb 效果略好于 xgb），使用两种算法分析实验数据，得到的结果有所不同。不知道是参数的原因还是其他什么原因
- 关于 logloss：官方定义的 logloss 和 logloss 的定义不一致，需要自己按照官方的公式定义 logloss
- 关于实验结果：全部实验数据都是线下测试，同时使用 lgb 和使用 xgb，效果不一，实验结果分析也可能不同
- 第一名队伍的解决方案涵盖了常用的机器学习技术，值得当作范例多加研究学习

## 4 总结和展望

本文从 NLP 领域、统计学领域、深度学习领域和安全背景出发，多角度分析阿里云提供的恶意软件数据集，管中窥豹，对比分析拥有复合背景能一定程度上提升模型效果和安全研究，存在的问题是特征的可解释性问题和跨领域结合的深度不够，虽然存在一定的问题，但是阻止不了机器学习在垂直领域越来越热的应用趋势。如有不妥之处，敬请指正，欢迎交流安全数据科学。

## 5 引用

https://tianchi.aliyun.com/competition/information.htm?raceId=231668

https://iami.xyz/AliSEC3/

https://hadxu.github.io/2018/09/25/%E7%AC%AC%E4%B8%89%E5%B1%8A%E9%98%BF%E9%87%8C%E4%BA%91%E5%AE%89%E5%85%A8%E7%AE%97%E6%B3%95%E5%A4%A7%E8%B5%9B%E5%88%9D%E8%B5%9B7th%E6%96%B9%E6%A1%88/

https://github.com/poteman/Alibaba-3rd-Security-Algorithm-Challenge

https://www.zhihu.com/question/63883507/answer/227019715

https://github.com/beader/tianchi-3rd_security/tree/v0.2

https://tianchi.aliyun.com/forum/new_articleDetail.html?spm=5176.8366600.0.0.6e50311faIzxop&raceId=231668&postsId=7220