Microsoft Malware Prediction Top 5%

4o4notfound / 2019-01-19 09:15:00 / 浏览数 5423 技术文章 技术文章 顶(0) 踩(0)

前言

目前最好成绩45/1039，暂时进入Top
5%，到现在为止，我觉得本次比赛的难点在于：一堆类别型属性，包括大量高势集、关键特征缺失、训练集和测试集的差异性。

0x01 数据再理解

一堆高势集

此次比赛数据集总共有82个属性，真正的数值型属性只有少数几个，比如

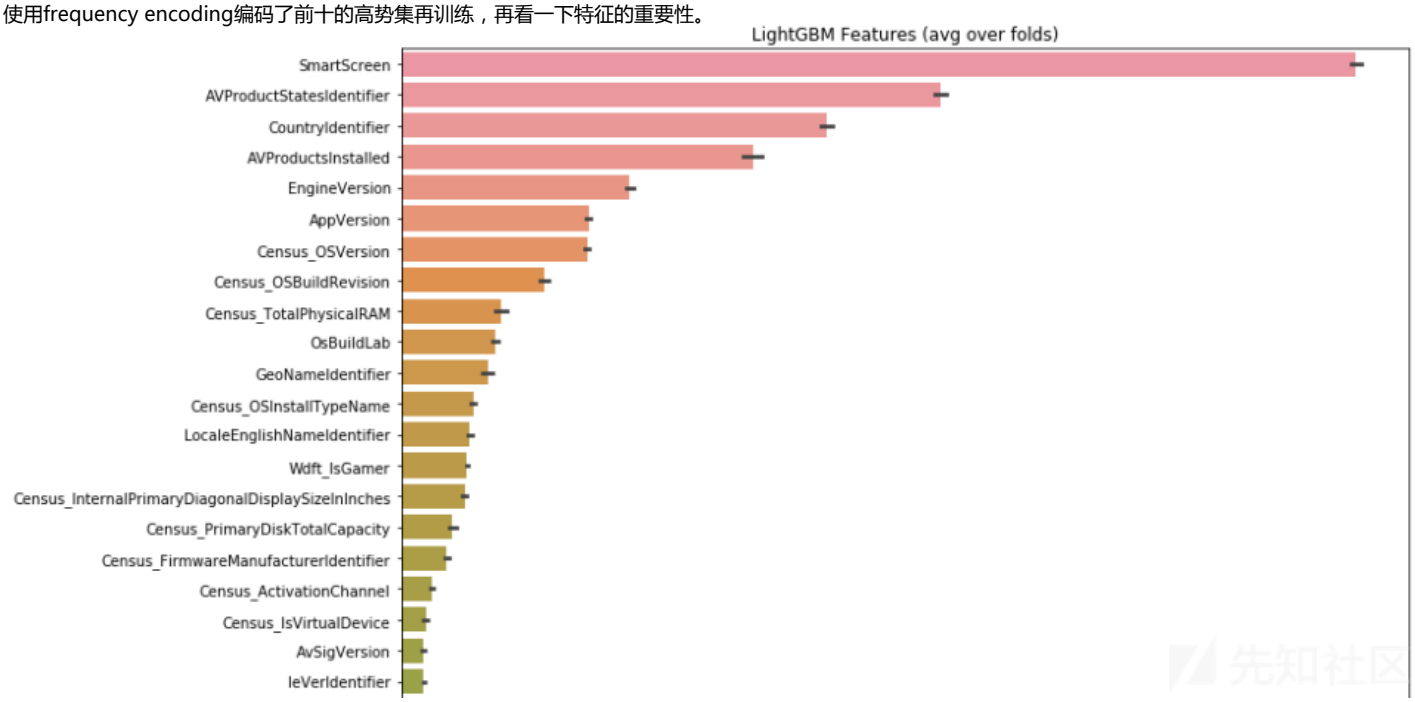
```
true_numerical_columns = [
'Census_ProcessorCoreCount',
'Census_PrimaryDiskTotalCapacity',
'Census_SystemVolumeTotalCapacity',
'Census_TotalPhysicalRAM',
'Census_InternalPrimaryDiagonalDisplaySizeInInches',
'Census_InternalPrimaryDisplayResolutionHorizontal',
'Census_InternalPrimaryDisplayResolutionVertical',
'Census_InternalBatteryNumberOfCharges'
]
```

这其中的属性有的也可以看做类别数据，所以数值属性就更稀少了，绝大多数都是类别型数据，而且有很多属性有上千上万个不同的值，这就叫做高势集，高势集很难处理。
encoding、Count encoding、LabelCount encoding、Hash
encoding。但是这几种类别特征编码的方式都很难处理高势集。One-hot编码很容易产生大量的稀疏特征，如果一个属性有十万个不同的值，那么经过One-hot编码会产生
encoding本身有缺点，数字之间的相邻性不足以代表类别数据之间的紧密相关性，所以大量采用Label encoding编码高势集也不现实。

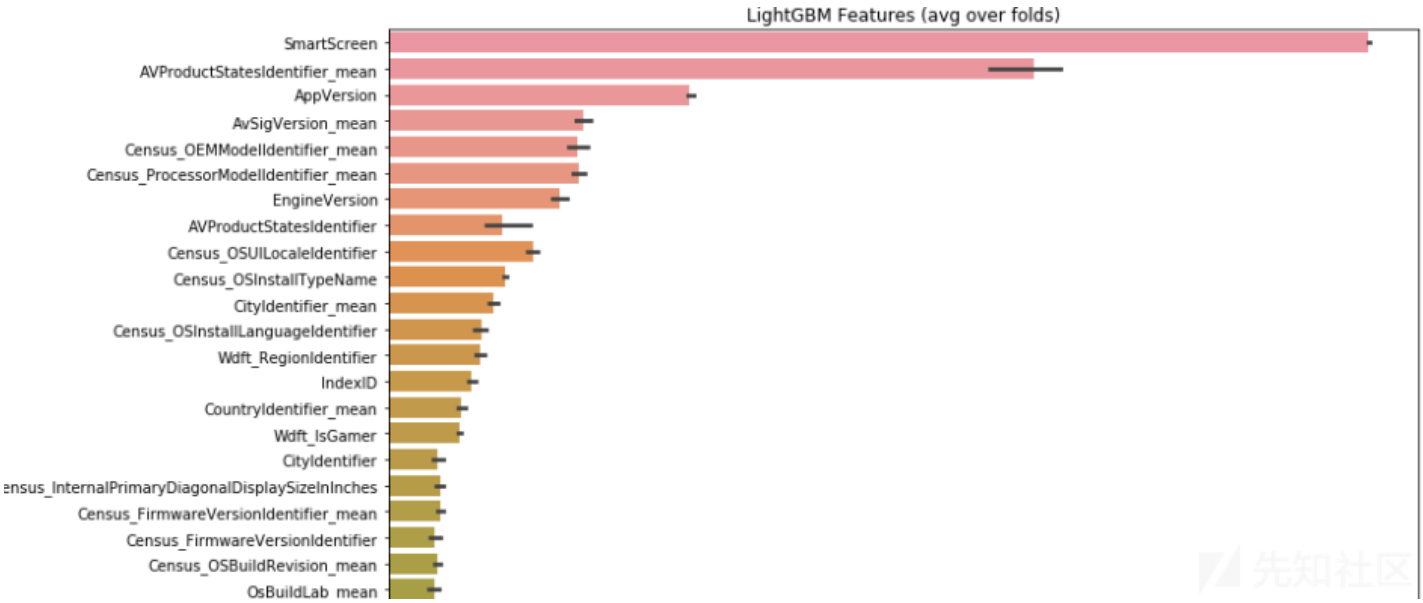
可能的关键特征

什么样的机器容易被感染？不具备领域知识，很直观的想，机器自身的防护状态越差可能越容易被感染。那么AppVersion，EngineVersion，AvSigVersion这三个防护状态
直接使用原始特征进行训练，训练完毕看一下特征的重要性，这三个特征都处于前十。

使用frequency encoding编码了前十的高势集再训练，再看一下特征的重要性。

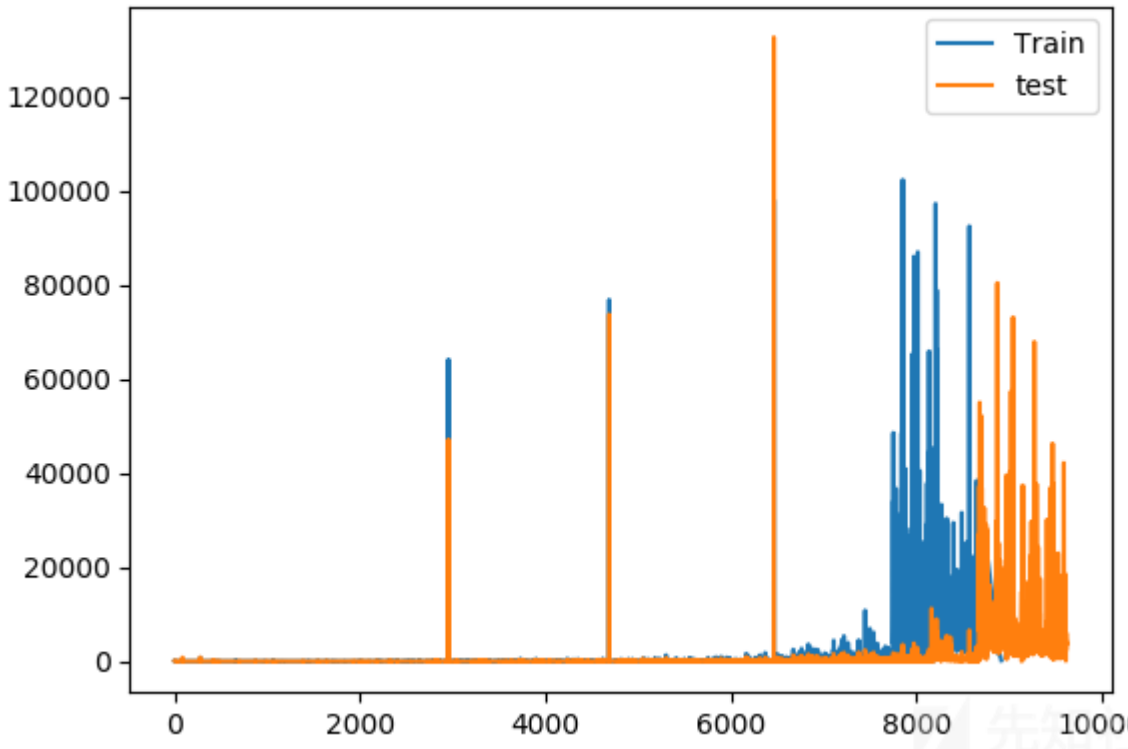


可以看出还是比较符合我们的认知，AppVersion、EngineVersion和AvSigVersion较为重要。
采用其他编码方式，这三个属性仍然很重要。



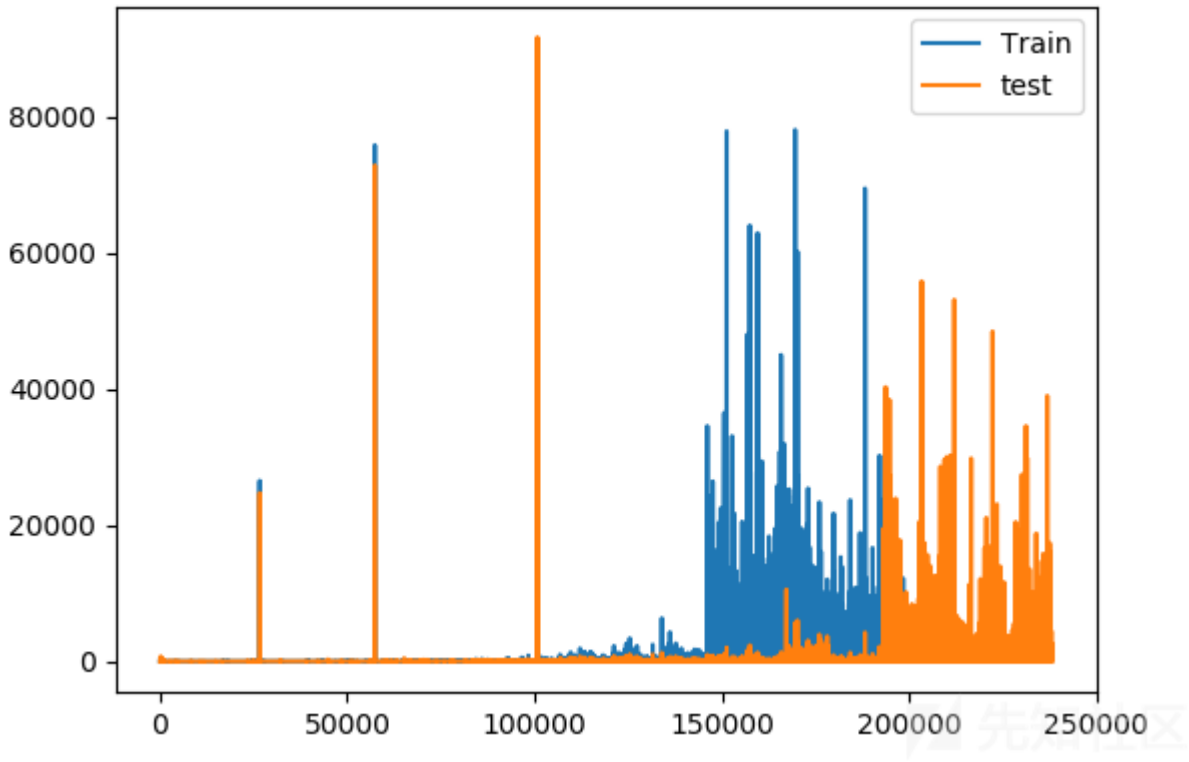
训练集和测试集的分布差异性

把数据按AvSigVersion排序，观察一下AvSigVersion在训练集和测试集中的分布，



横坐标是AvSigVersion的序列，纵坐标是每个AvSigVersion序列的个数，可以看出训练集和测试集存在明显的差别，测试集中有一些AvSigVersion从来没有在训练集中出现

把数据按AvSigVersion和AppVersion排序，观察一下AvSigVersion和AppVersion在训练集和测试集的分布，



很明显，训练集和测试集在AvSigVersion，AppVersion上的分布差异较大，测试集的版本数据大多较新，看到了一点预测未来的意思。

0x02 编码高势集

最初我是用frequency encoding编码高势集，效果较oral feature有了0.003左右的提升，之后开始使用mean encoding编码。关于mean encoding可以看这篇文章，写的很全面细致。使用平均目标值（以及其他目标统计数据）编码分类变量是处理高基数特征的常用方法，对于基于树的模型尤其有用。方法很encoding有一个缺点是存在过拟合，所以要加入一些噪声处理。这里我直接采用了Kaggle上一段代码。

```
def add_noise(series, noise_level):
   return series * (1 + noise_level * np.random.randn(len(series)))

def target_encode(trn_series=None,
   tst_series=None,
   target=None,
   min_samples_leaf=1,
   smoothing=1,
   noise_level=0):
   """
   Smoothing is computed like in the following paper by Daniele Micci-Barreca
   https://kaggle2.blob.core.windows.net/forum-message-attachments/225952/7441/high%20cardinality%20categoricals.pdf
   trn_series : training categorical feature as a pd.Series
   tst_series : test categorical feature as a pd.Series
   target : target data as a pd.Series
   min_samples_leaf (int) : minimum samples to take category average into account
   smoothing (int) : smoothing effect to balance categorical average vs prior
   """
   assert len(trn_series) == len(target)
   assert trn_series.name == tst_series.name
   temp = pd.concat([trn_series, target], axis=1)
   # Compute target mean
   averages = temp.groupby(by=trn_series.name)[target.name].agg(["mean", "count"])
   # Compute smoothing
   smoothing = 1 / (1 + np.exp(-(averages["count"] - min_samples_leaf) / smoothing))
   # Apply average function to all target data
   prior = target.mean()
   # The bigger the count the less full_avg is taken into account
   averages[target.name] = prior * (1 - smoothing) + averages["mean"] * smoothing
   averages.drop(["mean", "count"], axis=1, inplace=True)
   # Apply averages to trn and tst series
   ft_trn_series = pd.merge(
   trn_series.to_frame(trn_series.name),
```

```
   averages.reset_index().rename(columns={'index': target.name, target.name: 'average'}),
   on=trn_series.name,
   how='left')['average'].rename(trn_series.name + '_mean').fillna(prior)
   # pd.merge does not keep the index so restore it
   ft_trn_series.index = trn_series.index
   ft_tst_series = pd.merge(
   tst_series.to_frame(tst_series.name),
   averages.reset_index().rename(columns={'index': target.name, target.name: 'average'}),
   on=tst_series.name,
   how='left')['average'].rename(trn_series.name + '_mean').fillna(prior)
   # pd.merge does not keep the index so restore it
   ft_tst_series.index = tst_series.index
   return add_noise(ft_trn_series, noise_level), add_noise(ft_tst_series, noise_level)

trn, sub = target_encode(train[column],
                         test[column],
                         target=target,
                         min_samples_leaf=100,
                         smoothing=10,
                         noise_level=0.01)
```

开始使用mean encoding编码了前八个高势集，效果较frequency encoding有了0.002左右的提升，之后我扩大了编码范围（8-12-16-20），

```
mean_encoded_variables = [
'Census_OEMModelIdentifier',
'CityIdentifier',
'Census_FirmwareVersionIdentifier',
'AvSigVersion',
'Census_ProcessorModelIdentifier',
'Census_OEMNameIdentifier',
'CountryIdentifier',
'AVProductStatesIdentifier',
'DefaultBrowsersIdentifier',
'Census_FirmwareManufacturerIdentifier',
'Census_OSVersion',
'OsBuildLab',
'IeVerIdentifier',
'Census_OSBuildRevision',
'GeoNameIdentifier',
'LocaleEnglishNameIdentifier'
]
```

实验数据表明编码前16个高势集效果较优，又有了0.003左右的提升。还没确定只是简单扩大了特征范围让模型性能提升了，还是扩大的特征集合中正好覆盖了可能的关键特

0x03 构造新特征和特征选择

对我来说，打一场比赛大多靠的是体力，其次才是靠创造力。但是靠体力也可以学到很多，尝试各种想法，相当于靠这一场比赛活络了自己的知识库。构造新特征比较依赖

```
IndexID = (pd.concat([train[['MachineIdentifier',
'AppVersion',
'EngineVersion',
'AvSigVersion',
'HasDetections']],
 test[['MachineIdentifier',
'AppVersion',
'EngineVersion',
'AvSigVersion']]],
axis=0, sort=False)
  .reset_index(drop=True)
  .sort_values(['AppVersion', 'EngineVersion','AvSigVersion'])
  .reset_index(drop=True))

IndexID = pd.merge(IndexID, (IndexID[[ 'AppVersion',
 'EngineVersion',
 'AvSigVersion']].drop_duplicates()
.reset_index(drop=True)
.reset_index()
.rename({'index':'IndexID'}, axis=1)),
  on=[ 'AppVersion',
  'EngineVersion',
```

```
  'AvSigVersion'], how='left')
train['IndexID'] = (IndexID[IndexID.HasDetections.notnull()]
.sort_values(['MachineIdentifier'])
.reset_index(drop=True))['IndexID']

test['IndexID'] = (IndexID[IndexID.HasDetections.isnull()]
  .sort_values(['MachineIdentifier'])
  .reset_index(drop=True))['IndexID']

train = train.sort_values(['IndexID']).reset_index(drop=True)
```

增加这个特征，实验结果提升了0.001，说明还是有一点作用的。我尝试改变AppVersion、EngineVersion、AvSigVersion三者之间的顺序，5折交叉中，每种顺序在每折中

0x04 缩小训练集和测试集的GAP

到目前为止我的GAP还在0.05+，大佬们的GAP在0.04+，这也许就是我的single model LB：0688，multi model LB：0.692，排在第45名，大佬们的single model LB：0.699，multi model LB：0.701
排在第一名的原因。尝试了一些想法效果都不好，还是没找到有效处理GAP的方法，难道真要先做一个对抗性验证找到一个和测试集相似的训练集的子集，再用这子集训练叫kernel其中就用了训练集的子集去训练。还是得向Kaggler学习，尝试各种想法，比如尝试先结合特征再进行编码，围绕可能的关键特征构造新特征，虽然不一定有效果。

0x05 Reference

https://iami.xyz/MeiTuanMachineLearning-FeatureEnginne-Note/

https://www.kaggle.com/vprokopev/mean-likelihood-encodings-a-comprehensive-study

https://www.kaggle.com/c/microsoft-malware-prediction/discussion/77670

上一篇：利用JA3和JA3S实现TLS指纹识别 下一篇：如何将Xerosploit移植到M...

1. 0 条回复
   - 动动手指，沙发就是你的了！