1 背景

恶意应用程序的数量和多样性不断增长，传统的防御手段在很大的程度上无法发挥作用。尝试使用机器学习的方法，自动识别恶意软件的典型模式。

2 drebin 数据集介绍

1. feature_vectors 中包含 12 万个以 app 编号命名的文本文件，每一个文本文件中记录了对应 app 的相关属性，每一个属性都对应于 8 个类别（S1-S8）

source	feature sets	prefix of string
manifest	S_1 Hardware components	feature
	S_2 Requested permissions	permission
	S_3 App components	activity/service_receiver /provider/service
	S_4 Filtered intents	intent
disassembled code	S_5 Restricted API calls	api_call
	S_6 Used permissions	real_permission
	S_7 Suspicious API calls	call
	S_8 Network addresses	url

2. sha256_familiy.csv 中记录了所有恶意 app 的编号及其对应的家族，一共有 5560 个恶意 app，归属于 179 个家族。

3 恶意软件检测问题

3.1 检测目标

恶意软件（1）和正常软件（0）

3.2 数据集预处理

drebin 数据集中共有 129k+个 app 的相关记录，其中恶意 app 有 5560 个，非恶意 app 有 120k+个。从数据集中随机选择 5560 个恶意 app 以及 5560 个非恶意 app，设置训练集的比例为 70%，即训练集包含 7784 个 app 记录，测试集包含 3336 个 app 记录。

3.3 评估指标

查准率（precision）、查全率（recall）、F1 值、ROC 曲线与 AUC

3.4 算法介绍

- 逻辑回归（logistic regression）：是一种广义线性模型，常用于解决二分类问题，这种算法简单易行且计算迅速，可以作为一种基线方法与其他方法进行对比。
- 多层感知机（multilayer perceptron）：是一种最简单的神经网络模型，实验对 alpha、activation 和 solver 这三个参数进行了调优。
- 支持向量机（support vector machine）：其基本思想是求解能够正确划分训练数据集并且是几何间隔最大的分离超平面，实验中使用了默认参数获得了最

优性能。

- 随机森林（random forest）：是一个包含多个决策树的分类器，实验对 n_estimators 和 max_features 这两个参数进行了调优。
- 多项式朴素贝叶斯（multinomial naïve bayes）：是三种常见的朴素贝叶斯模型之。多项式模型和伯努利模型都适用于特征是离散的时候，不同的是伯努利模型中的特征的取值只能是 0 或 1；另外，高斯模型适用于特征变量是连续的时候。实验中使用了多项式模型的默认参数获得了最优性能。
- 决策树（decision tree）：这种方法创建了一种从数据特征中学习简单的决策规则的模型。通常用在集成方法中，构建随机森林模型。
- k 近邻方法（k-nearest neighbors）：是一种常用的监督学习方法，与前面的方法不同的是这是一种懒惰学习方法，在训练阶段仅仅是把样本保存起来，收到测试样本后再进行处理。实验中我们对 n_neighbors、weights 和 algorithm 这三个参数进行了调优。

3.5 特征介绍

- S_1 Hardware components：应用运行中请求访问的硬件设备。可以访问 GPS 和网络模块的应用程序可以搜集用户的位置数据，并通过网络发送给攻击者。
- S_2 Requested permissions：应用程序安装时需要由用户授予的权限。当前恶意软件中有很多都需要发送高级 SMS 消息，从而请求 SEND_SMS 权限。
- S_3 APP components：应用程序中使用的组件，这些组件定义了与系统的不同接口。恶意软件族 DroidKungFu 共享特定的组件。
- S_4 Filtered intents：android 上的进程间和进程内通信主要通过 intent 执行。

恶意软件中将 intent 消息 BOOT_COMPLETED 用于在重启手机后立即触发恶意活动。

- S_5 Restricted API calls：android 权限系统限制了对一系列关键 API 调用的访问。如果使用了受限的 API 调用而没有请求相应的权限，则表明恶意软件可能在使用 root 漏洞。
- S_6 Used permissions：将 API 调用和对应的权限匹配起来，提取出既请求又使用过的权限集合。
- S_7 Suspicious API calls：恶意软件中经常出现的允许访问敏感数据和资源的 API 调用。
- S_8 Network addresses：反汇编代码中出现的所有 IP 地址、主机名和 URL。恶意软件会定期建立网络连接以获取命令或泄露从手机上搜集的信息。

3.6 实验结果与分析

3.6.1 模型对比实验

表 1 七种模型的查准率、查全率和 F1 对比

	Precision	Recall	F1 score
Logistic regression	0.86	0.78	0.82
MLP	0.88	0.91	0.90
SVM	0.94	0.89	0.91
Random forest	0.93	0.94	0.94
Decision tree	0.90	0.93	0.92

MultinomialNB	0.65	0.83	0.73
KNN	0.91	0.95	0.93

在这七个模型中，多项式朴素贝叶斯模型作为一种基线方法其性能最差，查准率和 F1 值都最低，是唯一一个 F1 值没有达到 0.80 的模型。由于朴素贝叶斯模型使用了“属性条件独立性假设”，所以如果样本属性有关联时其效果不好。因此 drebin 数据集中所收集的 app 的属性之间具有较强关联（例如特征 5 是安卓系统被限制的一系列重要的 API 调用，而特征 6 使用的权限，与特征 5 中的 API 调用具有紧密的匹配关系）可能是多项式模型表现最差的主要原因。逻辑回归是性能第二差的模型，其 F1 值和查全率都最低，F1 值只有 0.82 而其余 5 种都超过了 0.90。MLP 和 SVM 这两种方法性能相似，MLP 的 F1 值比 SVM 的 F1 值略低 0.01，而查准率 SVM 略高，查全率 MLP 略高。决策树模型和 k 近邻模型性能相似，其中 k 近邻方法在查全率、查准率和 F1 值上略高，是仅次于随机森林模型的方法。KNN 算法适用于特征较少且数据集稠密的情况，因此在 drebin 数据集上取得了不错的效果。随机森林的表现最好，F1 值最高为 0.94，而查准率和查全率仅比最高值低 0.01。随机森林这一模型包含决策树算法的优点，这一方法可以从数值上计算特征的重要性从而做出更好的特征选择，另外它还弥补了决策树容易出现过拟合的缺陷。

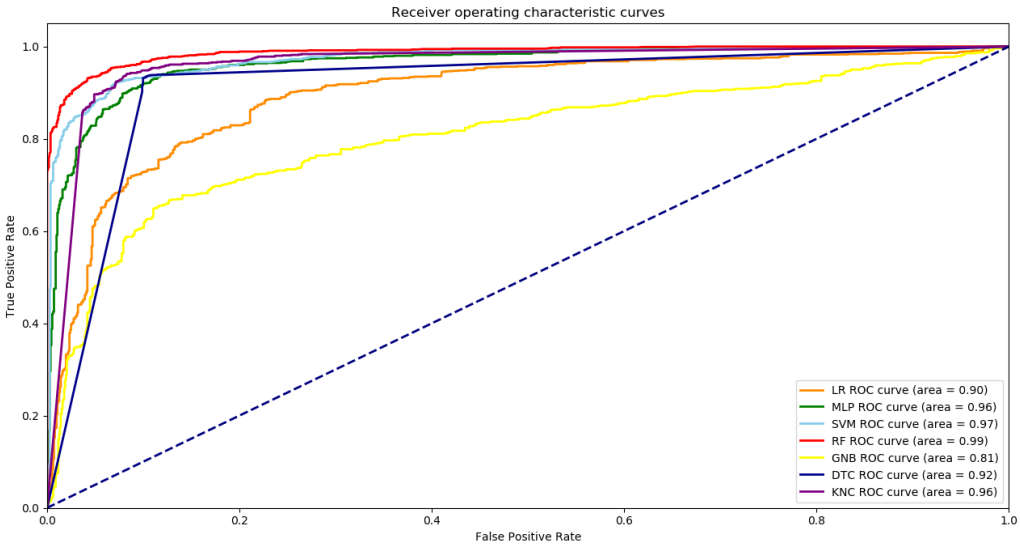

图 1 七种模型的 ROC 曲线和对应的 AUC

在这七种模型的 ROC 曲线图中，可以明显看出多项式朴素贝叶斯模型和逻辑回归对应的 ROC 曲线被包在另外五种模型的 ROC 曲线中，这意味着多项式朴素贝叶斯和逻辑回归的性能最差，其 AUC 值分别等于 0.81 和 0.90。决策树模型的 AUC 值为 0.92，而 MLP、SVM 和 K 近邻模型的 AUC 值相近，分别为 0.96、0.97 和 0.96。随机森林的 AUC 值为 0.99，这表明随机森林这个模型的性能最佳。

3.6.2 特征重要性

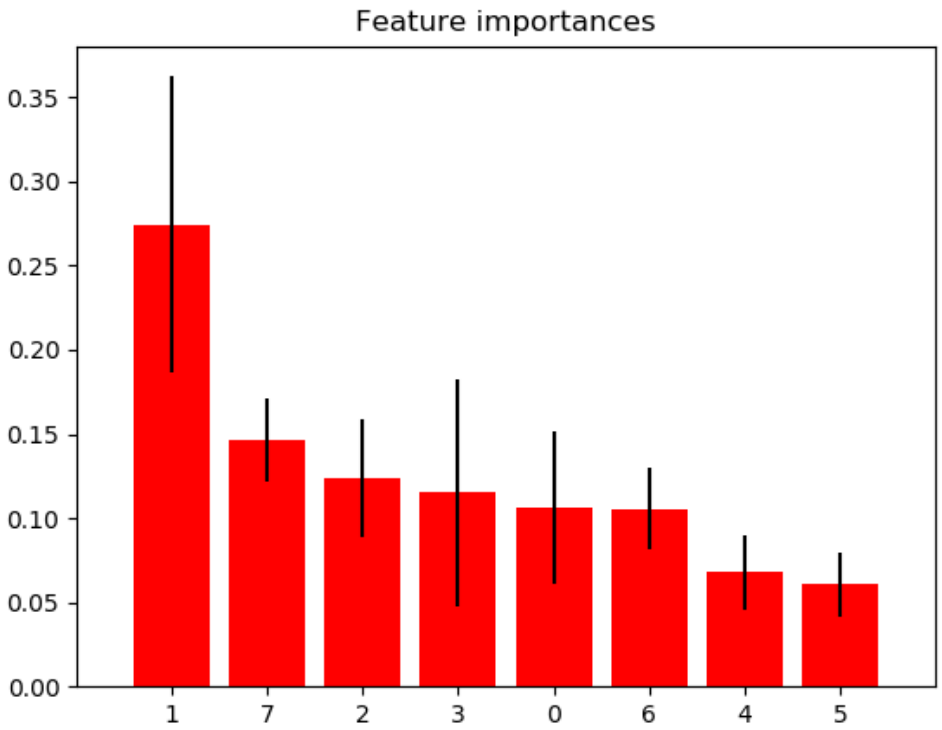

图 2 随机森林模型中特征重要性

如图 2 所示，特征 1 和 7 拥有最高的重要性，这两个特征分别对应的软件请求权限的次数和网络地址的数量这两个特征。这意味着如果一个 app 更频繁地请求系统权限并且需要建立更多的网络连接，那么这个 app 就更可能是恶意软件。

3.6.3 特征分析

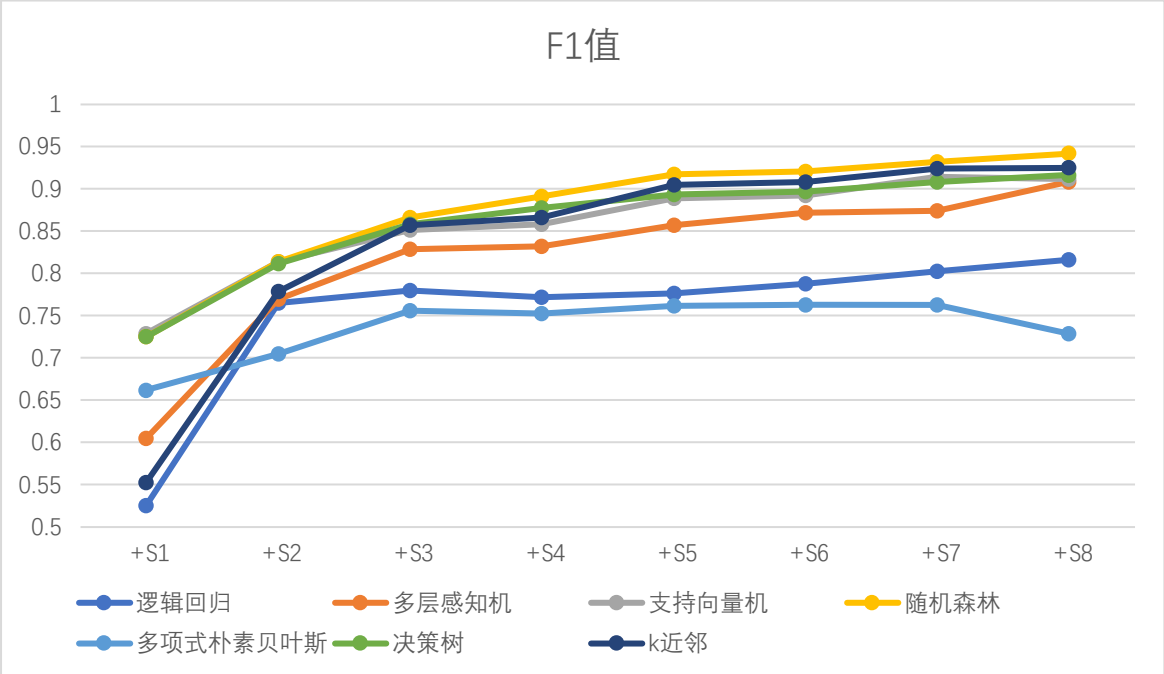

图 3 逐步添加特征 S1-S8，7 个模型的 F1 值的变化

如图 3 所示，在只考虑特征 1 的情况下，这七中模型的 F1 值均超过了 0.5。在其余 7 个特征中，当加入特征 2 和特征 3 后七种模型的 F1 值都有较大的上升，这表明特征 2 和特征 3 可以更有效地描述恶意软件的特征。当加入特征 5 和特征 6 后七种模型的 F1 值都略有增加，而这两个特征在随机森林模型中是特征重要性最低的两个特征。这意味着尽管一个特征在随机森林模型中重要性较低，这个特征在其他模型中也可能会有更积极的影响。当加入特征 4 后，逻辑回归和多项式朴素贝叶斯模型的 F1 值略微下降，而在其他五种模型上均产生积极影响。这表明逻辑回归和多项式朴素贝叶斯在此任务上表现不好的共同原因是，相较于另外五种模型，它们不能很好的利用特征 4 所包含的信息对恶意软件进行判断。当加入特征 7 和特征 8 后，多项式朴素贝叶斯的 F1 值大幅下降，而

其余 6 种模型的 F1 值均有上升。尤其是在特征 8 加入后，朴素贝叶斯的性能收到很大的负面影响，而这一特征在性能最优的随机森林模型中的重要性位列第二，这一特征的影响直观地解释了朴素贝叶斯具有最差性能的主要原因是不能很好利用特征 8 中信息来进行恶意软件的识别。

表 2 随机森林中的特征重要性

特征	重要性	特征	重要性
S_2 Requested permissions	0.271183	S_1 Hardware components	0.105809
S_8 Network addresses	0.147707	S_7 Suspicious API calls	0.105378
S_3 APP components	0.124666	S_5 Restricted API calls	0.066397
S_4 Filtered intents	0.111902	S_6 Used permissions	0.061069

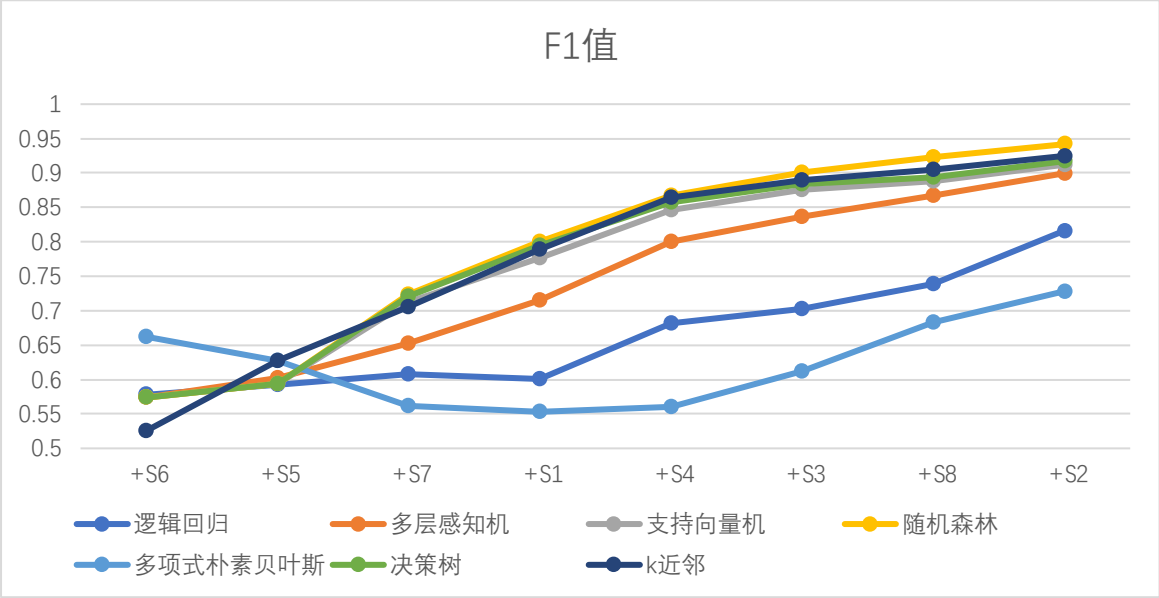

图 4 按照随机森林模型中特征重要性由低到高逐步加入特征

如表 2 所示，按随机森林中的特征重要性大小排序，由低到高，分别是特征S_6、特征S_5、特征S_7、特征S_1、特征S_4、特征S_3、特征S_8和特征S_2。对比图 4 和图 3，首先，可以看出改变加入特征的顺序对于模型最

终取得的 F1 值没有影响。k 近邻、支持向量机、决策树、多层感知机和随机森林这五种方法的变化趋势十分相似。其中，在加入特征 S7、S1、S4 和 S3 时，多层感知机的增幅略低于另外四种模型。k 近邻方法在只有特征 S6 时性能最差，而当加入特征 S5 后，其 F1 值的增幅比另 4 种模型都大。逻辑回归模型的 F1 值随着逐步加入特征，整体趋势仍为增加，只是在加入特征 S1 时略有降低。多项式朴素贝叶斯模型的 F1 值的整体趋势是先降低再增加，当加入特征 S5、S7 和 S1 时，F1 值先降低；然后随着加入特征 S4、S3、S8 和 S2，F1 值逐渐升高。值得注意的是，与图 1 的曲线对比，特征 S5、特征 S4 和特征 S8 对多项式朴素贝叶斯模型的 F1 值的影响正好相反。在第一种顺序中，加入特征 S5 后 F1 值略有上升，加入特征 S8 和 S4 后 F1 值略有降低；而在第二种顺序中，加入特征 S5 后有较大幅度的降低，加入特征 S8 和 S4 后 F1 值得到提升。当特征加入的顺序发生变化后，在多项式朴素贝叶斯模型中，每一个特征对其 F1 值的影响表现出不稳定的现象，这也可能暗示着多项式朴素贝叶斯模型不适合此任务。

4 类别不平衡问题

4.1 解决方案

drebin 数据集中包括恶意软件 5560 个，非恶意软件 12 万多个。为了解决 drebin 数据集上出现的类别不平衡问题，我们主要从两个方面来采取策略：（1）设计损失函数，为数据量更少的类别数据（恶意软件）分配更大的权重，这样在

模型训练的过程中，模型会更多地关注这类数据；（2）对原数据集进行重采样，包括过采样和欠采样。

4.2 实验结果

在第一个策略中，我们实现了 focal loss，并且和 cross entropy 损失函数进行对比。实验结果如下表 3。

表 3 设计新的损失函数解决类别不平衡问题

NO.	超参数	训练集	测试集
1	repeat_sampling_times=8 layers_dims = (8, 16, 8, 4, 1) optimizer = 'momentum' beta = 0.9 learning_rate = 0.1 num_epochs = 10000 loss_type = 'focal loss' mini_batch_size = 4096 no regularization	Precision: 0.87615 Recall: 0.84829 F1 score: 0.86199 Accuracy: 0.9279	Precision: 0.8733 Recall: 0.84738 F1 score: 0.86015 Accuracy: 0.92739
2	repeat_sampling_times=8 layers_dims = (8, 16, 8, 4, 1) optimizer = 'momentum' beta = 0.9 learning_rate = 0.1 num_epochs = 8800 loss_type = 'cross entropy' mini_batch_size = 4096 no regularization	Precision: 0.86823 Recall: 0.85893 F1 score: 0.86355 Accuracy: 0.92795	Precision: 0.86617 Recall: 0.85853 F1 score: 0.86233 Accuracy: 0.92777

从表 3 的实验数据中，我们可以看出采用了 focal loss 后，模型的性能并没有取得太大进步，甚至还变差了一些。我们给出两个实验训练过程的损失变化曲线，可以看出一点原因。

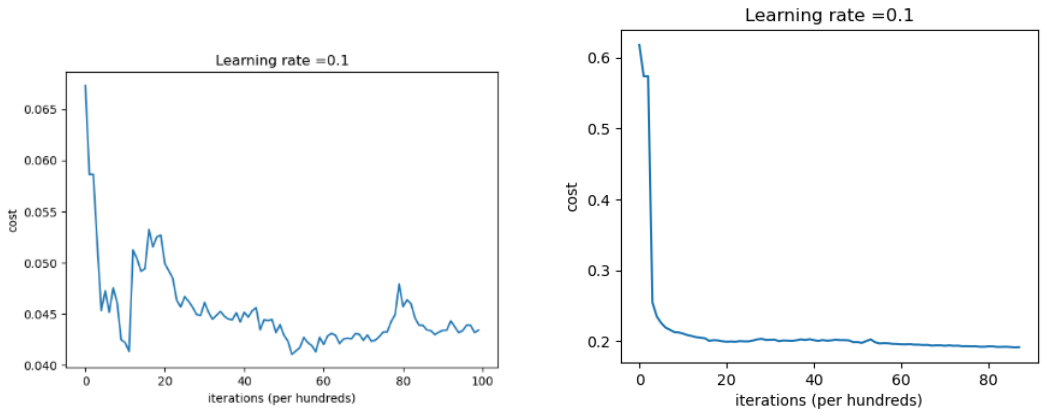

图 5 采用 focal loss（左）和 entropy loss（右）在训练过程中损失值变化曲线图。

采用 focal loss 后模型训练中，损失值在下降过程中会有较大幅度的振荡，而 entropy loss 在训练后期，损失值变化很平稳。这导致采用 focal loss 训练模型，取不同的 num_epochs 对模型性能的影响较大，这里取 10000 次 num_epochs 可以看出此时的损失值并不是训练中最低的值，说明此时模型性能可能还不是最优的。

在第二个策略中，我们尝试了 imblearn 包中提供的各种重采样方法，进行了相关实验。下表给出了采用过采样、欠采样和过采样结合欠采样三类方法，进行调参后在训练集和测试集上的实验结果。

表 4 采用重采样策略解决类别不平衡问题

No.	超参数	训练集性能	测试集性能
1	过采样策略：RandomOverSampler x_resampled shape: (246906, 8) y_resampled shape: (246906, 1) num of positive samples: 123453 layers_dims = (8, 64, 32, 8, 1) optimizer = 'momentum' beta = 0.9 learning_rate = 0.1 num_epochs = 1600 1700 次后 cost 为 nan	Precision: 0.88348 Recall: 0.84905 F1 score: 0.86592 Accuracy: 0.86852	Precision: 0.88589 Recall: 0.84971 F1 score: 0.86742 Accuracy: 0.87015

	lambd = 0.03 mini_batch_size = 2**16		
2	过采样策略：SMOTE x_resampled shape: (246906, 8) y_resampled shape: (246906, 1) num of positive samples: 123453 layers_dims = (8, 64, 32, 8, 1) optimizer = 'momentum' beta = 0.9 learning_rate = 0.1 num_epochs = 1400 1500 次后 cost 为 nan lambd = 0.03 mini_batch_size = 2**16	Precision: 0.83972 Recall: 0.77064 F1 score: 0.8037 Accuracy: 0.81176	Precision: 0.83851 Recall: 0.77404 F1 score: 0.80499 Accuracy: 0.81252
3	欠采样策略：RandomUnderSampler x_resampled shape: (11120, 8) y_resampled shape: (11120, 1) num of positive samples: 5560 layers_dims = (8, 64, 32, 8, 1) optimizer = 'momentum' beta = 0.9 learning_rate = 0.1 num_epochs = 10900 11000 次后 cost 为 nan lambd = 0.03 mini_batch_size = 2**16	Precision: 0.95667 Recall: 0.95176 F1 score: 0.95421 Accuracy: 0.95427	Precision: 0.90691 Recall: 0.908 F1 score: 0.90745 Accuracy: 0.90767
4	欠采样策略：RandomUnderSampler x_resampled shape: (11120, 8) y_resampled shape: (11120, 1) num of positive samples: 5560 layers_dims = (8, 64, 32, 8, 1) optimizer = 'momentum' beta = 0.9 learning_rate = 0.1 num_epochs = 2200 2300 次后 cost 为 nan lambd = 0.1 mini_batch_size = 1024	Precision: 0.96043 Recall: 0.96536 F1 score: 0.96289 Accuracy: 0.96274	Precision: 0.90685 Recall: 0.9074 F1 score: 0.90712 Accuracy: 0.90737
5	欠采样策略：RandomUnderSampler x_resampled shape: (11120, 8) y_resampled shape: (11120, 1)	Precision: 0.96146 Recall: 0.96664 F1 score: 0.96404	Precision: 0.9121 Recall: 0.911 F1 score: 0.91155

	num of positive samples: 5560 layers_dims = (8, 64, 32, 8, 1) optimizer = 'momentum' beta = 0.9 learning_rate = 0.1 num_epochs = 2200 2300 次后 cost 为 nan lambd = 0.3 mini_batch_size = 1024	Accuracy: 0.9639	Accuracy: 0.91187
6	欠采样策略：RandomUnderSampler x_resampled shape: (11120, 8) y_resampled shape: (11120, 1) num of positive samples: 5560 layers_dims = (8, 64, 32, 8, 1) optimizer = 'momentum' beta = 0.9 learning_rate = 0.1 num_epochs = 5700 5800 次后 cost 为 nan lambd = 1 mini_batch_size = 1024	Precision: 0.9713 Recall: 0.98127 F1 score: 0.97626 Accuracy: 0.9761	Precision: 0.89697 Recall: 0.92664 F1 score: 0.91156 Accuracy: 0.91037
7	欠采样策略：RandomUnderSampler x_resampled shape: (11120, 8) y_resampled shape: (11120, 1) num of positive samples: 5560 layers_dims = (8, 64, 32, 8, 1) optimizer = 'momentum' beta = 0.9 learning_rate = 0.1 num_epochs = 1300 1400 次后 cost 为 nan lambd = 1 mini_batch_size = 1024	Precision: 0.91097 Recall: 0.93739 F1 score: 0.92399 Accuracy: 0.92279	Precision: 0.89374 Recall: 0.9104 F1 score: 0.902 Accuracy: 0.90138
8	过采样与欠采样结合：SMOTETomek x_resampled shape: (246830, 8) y_resampled shape: (246830, 1) num of positive samples: 123415 layers_dims = (8, 16, 8, 4, 1) optimizer = 'momentum' beta = 0.9 learning_rate = 0.1	Precision: 0.87419 Recall: 0.83361 F1 score: 0.85342 Accuracy: 0.85691	Precision: 0.87579 Recall: 0.8366 F1 score: 0.85574 Accuracy: 0.85878

	num_epochs = 6400 6500 次后 cost 为 nan lambd = 0.1 mini_batch_size = x_train.shape[1]		

在 imblearn 包中提供了各种过采样和欠采样的方法，上面表格中给出了我们尝试在各种采样方法下调参并且训练收敛的神经网络模型，还有大部分实验训练的模型均未收敛。下面给出一些从上述实验数据中得到的观察：

（1） 实验 1、2、8 采用的过采样策略（RandomOverSample 和 SMOTE）和过采样与欠采样结合策略（SMOTEomek）将原数据集（大约 13 万条数据）采样成正负样例均衡的新数据集（大约 24 万条数据）。这三个实验中由于训练集较大，训练时间更久，同时模型似乎更难收敛，三个模型的损失变化都表现为：初期训练很久 cost 不变，然后 cost 在 200epochs 内迅速下降，随后模型在 cost 数值计算时出现下溢情况（cost=nan），上表中给出的模型训练效果均是在出现 cost=nan 之前停止训练获得的模型参数。同时，上述三个实验中，我发现 mini_batch_size 越大越好，当 mini_batch_size 变小后，模型的 cost 会在还没下降之前就产生 cost=nan 的问题。

（2） 实验 3-7 只采用了欠采样策略 RandomUnderSample，原因是 imblearn 中提供的另一个欠采样方法 ClusterCentroids 会将原数据集中的特征向量（numpy.ndarray 对象）的数据类型从 numpy.int32 转换为 numpy.float64 从而导致 numpy.core 报错，因为内存原因不支持一个（5560，5560）大小的数据类型为 numpy.float64 的 ndarray。

RandomUnderSample 将原数据集采样成类别均衡的新数据集（11120 条数据），相比于过采样方法，数据量变小为 1/24。在这个相对更小的数据集上，我们构造的神经网络结构为（8, 64, 32, 8, 1），并调整超参取得了相

对更优的效果。如表 4，我们在实验 6 中取得最好的 F1 值 0.91156 和最高的 Recall=0.92664。同时我们观察到模型出现了比较严重的过拟合问题，因此从实验 3 到实验 6，我们逐渐增大了 L2 正则化参数 lambd 以减轻这一问题（使用 dropout 方法模型训练效果较差）。另外，在小数据集上，似乎更小的 mini_batch_size 运行的更好。

总结上述重采样实验结果，采用神经网络结构（8, 64, 32, 8, 1）在 24 万数据集上训练模型，这时模型很难训练成功会出现各种数值计算问题；而在 1 万大小的数据集上训练模型，模型的训练和预测效果不错，但是此时出现了严重的过拟合问题，采用 L2 正则化，可以有效降低模型对训练集的拟合精度，但没有有效提升测试集上的效果，这显然不是我们想要的。一个结构为（8, 64, 32, 8, 1）的神经网络模型，其模型容量相较于 1 万大小的数据集是足够大的，因此容易出现过拟合问题，但是当我们增大数据集的大小时，这个神经网络模型又变得难以训练了。