LEMNA: Explaining Deep Learning based Security Applications 论文翻译&笔记

ABSTRACT

尽管深度学习在各个领域都显示出巨大的潜力，但缺乏透明性限制了其在安全性或安全性至关重要的领域中的应用。 现有研究已经尝试开发解释技术以为每个分类决策提供可解释的说明。 不幸的是，当前的方法只针对非安全性任务（例如，图像分析）进行了优化（提供解释）。 他们的关键假设在安全应用程序中经常被违反，导致这些解释在安全领域方面的精确度很差。

在本文中，我们提出了LEMNA，一种专用于安全性应用程序的高精确解释方法。 给定输入数据样本，LEMNA会生成一小部分可解释的重要特征，以利用这些特征解释如何对输入样本进行分类。 **核心思想**是使用简单的可解释模型近似（逼近）复杂的深度学习**决策边界**的局部区域。 本地可解释模型是专门设计用于（1）处理好程序的**特征依赖**，以更好地与安全应用程序配合使用（例如二进制代码分析）；（2）处理**非线性局部边界**以提高解释的精确度。 我们使用了两个在安全方面流行的深度学习应用程序来评估我们的系统（一个恶意软件分类器和一个用于二进制逆向工程的函数开始位置检测器）。 广泛的评估表明，与现有方法相比，LEMNA的解释具有更高的精确度。 此外，我们演示了LEMNA的实际用例，以帮助机器学习开发人员验证模型行为，排除分类错误并自动修补目标模型的错误。

> 决策边界解析：https://blog.csdn.net/jodie123456/article/details/88870616

CCS CONCEPTS： Security and privacy → Software reverse engineering;
KEYWORDS： Explainable AI, Binary Analysis, Deep Recurrent Neural Networks

INTRODUCTION

近年来，深度神经网络在构建安全应用程序方面显示出巨大潜力。 到目前为止，研究人员已经成功地将深度神经网络用于训练分类器，以**进行恶意软件分类，二进制逆向工程和网络入侵检测**，所有这些都达到了极高的精度。

虽然安全性从业人员对高精度很感兴趣，但他们担心深度学习模型缺乏透明性，因此不愿在安全和关键安全领域广泛采用深度学习分类器。 更具体地说，深度神经网络可以轻松地包含数十万甚至数百万个神经元。 该网络一旦经过海量数据集的训练，便可以提供很高的分类精度。 但是，**网络的高度复杂性也导致模型的“可解释性”低**。 这种高度的复杂性使得我们很难理解神经网络的某些决策，从而导致了诸如无法信任神经网络以及无法有效判断神经网络的错误等问题。

为了提高深度神经网络的透明度，研究人员开始研究解释方法以解释分类结果。 现有的大多数作品都专注于非安全性应用，例如图像分析或自然语言处理（NLP）。 图1a显示了一个示例。 给定一个输入图像，解释方法通过将最有影响力的特征精确地指向最终决策来解释分类结果。 常见方法包

括在**网络中运行前向传播或后向传播以推断重要特征。** 在没有分类器详细信息的情况下，更高级的方法会在“黑盒子”设置下产生解释。 基本思想是**使用线性模型来推断重要决策特征，以近似局部决策边界。**

不幸的是，现有的解释方法并不直接适用于安全应用程序。 首先，大多数现有方法都是为图像分析而设计的，它更喜欢使用卷积神经网络（CNN）。 但是，CNN模型在安全领域不是很流行。 诸如二进制逆向工程和恶意软件分析之类的安全应用程序具有高度的函数依赖性（例如，二进制代码序列），或者需要高度的可伸缩性。然而，**循环神经网络（RNN）或多层感知器模型（MLP）被更广泛地使用于安全程序中。** 到目前为止，**在RNN上还没有一种很好的解释方法。** 其次，正如我们在§5中的实验所证实的那样，现有方法的解释精确度仍然很低。 这对于图像分析可能是可以接受的，但是会在安全应用程序中造成严重的麻烦。 例如，在图1a中，**突出显示的像素并不完全准确**（特别是在边缘区域），但是足以提供直观的理解。但是，对于诸如二进制分析，**错误地突出显示一个字节的代码可能会导致严重的误解或解释错误。**（提出了质疑，CNN的解释并不能应用于RNN/安全应用程序中来）

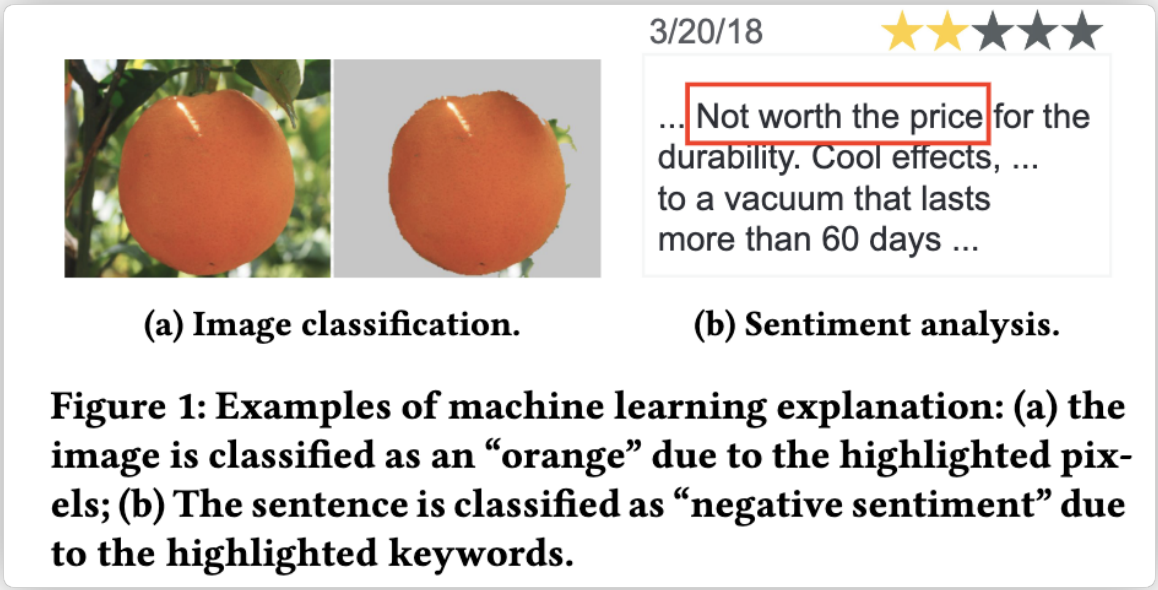


Figure 1: Examples of machine learning explanation: (a) the image is classified as an “orange” due to the highlighted pixels; (b) The sentence is classified as “negative sentiment” due to the highlighted keywords.

Our Designs。 在本文中，我们寻求开发一种专门用于安全应用程序的创新、高精确的解释方法。我们的方法在黑盒设置下工作，并引入了专门的设计来应对上述的挑战。 给定一个输入数据实例x和一个分类器（例如RNN），我们的方法**旨在识别一小组对x的分类有关键作用的要素。 这是通过生成目标分类器决策边界在x附近的局部近似值来完成的。** 为了显着提高近似的精确度，我们的方法不再假定局部检测边界是线性的，也不再假定特征是独立的。 相反，我们**借用混合回归模型来近似非线性的局部决策边界，同时我们通过fused lasso来加强解释精度。**这样一种设计一方面提供了足够的灵活性来优化对于非线性决策边界的拟合，**另外一方面fused lasso 可以很好的抓住不同特征之间的依赖性。**

我们的设计基于两个关键想法。 首先，从理论上讲，混合回归模型可以在给定足够数据的情况下近似线性和非线性决策边界。 这使我们能够灵活地优化非线性边界的局部逼近并避免较大的拟合误差。 其次，**“fused lasso”是通常用于捕获特征依赖性的惩罚性常用术语。** 通过将**fused lasso**添加到学习过程中，**混合回归模型可以将要素作为一个组，从而捕获相邻要素之间的依存关系。** 这样，我们的方法通过同时保留深度学习模型的局部非线性和特征相关性来产生高精确解释结果。 为了方便起见，我们将我们的方法称为“使用非线性近似的局部解释方法”或LEMNA。

Evaluations。为了证明我们的解释模型的有效性，我们将LEMNA应用于两个较好的安全应用程序：对PDF恶意软件进行分类，以及用于二进制逆向工程的函数开始位置的检测器。 **分类器分别接受10,000个PDF文件和2,200个二进制文件的训练，两者的准确性均达到98.6％或更高。** 我们使用LEMNA来解释其分类结果，并开发一系列精确度指标来评估这些解释的正确性。 **通过直接将近似的检测边界与真实边界进行比较，或者通过运行端到端特征测试来计算精确度指标。** 结果表明，LEMNA在所有不同分类器和应用程序设置上的性能均明显优于现有方法。

除了有效性评估之外，我们还将演示安全分析人员和机器学习开发人员如何从解释结果中受益。 首先，我们证明LEMNA可以通过解释分类器如何做出正确的决策来帮助建立信任。 特别是，**对于二进制分析和恶意软件分析，我们证明分类器已成功学习了各自领域中的许多知名启发式方法和“黄金法则”**。 其次，我们说明LEMNA可以从分类器中提取“新知识”。 这些新的启发式方法很难直接进行手动总结，但是一旦由LEMNA提取出来，对领域专家便具有直观意义。 最后，借助LEMNA的功能，分析师可以解释为什么分类器会产生错误。这使分析师可以通过增加针对每个可解释错误的训练样本自动生成目标补丁，并通过有针对性的重新训练来提高分类器的性能。

contributions Our paper makes three key contributions.:

- 我们设计和开发LEMNA，这是针对基于深度学习的安全应用程序的一种专门的解释方法。 **使用“fused lasso”增强的混合回归模型，LEMNA可以为包括RNN在内的一系列深度学习模型生成高精确解释结果。**
- 我们使用两种流行的安全应用程序评估LEMNA，其中包括PDF恶意软件分类和二进制逆向工程中的函数开始位置的检测。 **我们提出了一系列“精确度”指标来量化解释结果的准确性。** 我们的实验表明，LEMNA大大优于现有的解释方法。
- 我们演示了解释方法的实际应用。 对于二进制分析和恶意软件检测，LEMNA阐明了分类器为何做出正确和错误决定的原因。 **我们提供了一种简单的方法，可自动将见解转换为可操作的步骤，以修补分类器的目标错误**

据我们所知，这是第一个专门针对安全应用程序和RNN定制的解释系统。 我们的工作只是提高模型透明度以更有效地测试和调试深度学习模型的第一步。 通过使决策过程可解释，我们的努力可以为构建用于关键应用程序的可靠深度学习系统做出积极贡献。

EXPLAINABLE MACHINE LEARNING

在本节中，我们首先介绍可解释的机器学习的背景，然后讨论现有的解释技术。 随后，在第§3节中，我们将使用深度学习模型介绍关键的安全应用程序，并讨论为什么现有的解释技术不适用于安全应用程序。

Problem Definition

可解释的机器学习旨在为分类结果提供可解释的说明。 更具体地说，给定输入实例x和分类器C，分类器将在测试期间为x分配标签y。 然后，解释技术旨在说明为什么实例x被分类为y。 这通常涉及确定一组对分类过程（或结果）有重要贡献的重要特征。如果这些被选择的特征是可以理解的，那么我们就认为这些特征给出了一个解释。图1显示了图像分类和情感分析的示例。 可以通过所选功能（例如，突出显示的像素和关键字）来解释分类器决策。

在本文中，我们专注于深度神经网络（DNN）以开发用于安全应用程序的解释方法。 到目前为止，大多数现有的解释方法都是为图像分析或NLP设计的。 我们将它们分为“白盒”和“黑盒”方法，并描

述它们的工作方式。

Whitebox Explanation Methods（可略）

大多数现有的解释技术都在已知模型结构，参数和训练数据的白盒设置下工作。 这些技术也称为“深度解释方法”，主要针对CNN设计。 他们利用两种主要策略来推断特征的重要性：
（1）基于正向传播的输入或结构遮挡；
（2）基于梯度的反向传播。
我们将在下面讨论这些技术。

基于正向传播的方法。 给定输入样本，关键思想是干扰**输入**（或**隐藏的网络层**）并观察相应的变化。 一般来说，**干扰重要功能很可能导致网络结构和分类输出发生重大变化。** 现有方法要么使特征的子集无效，要么移除网络的中间部分。 最近的工作将该思想扩展到**检测对抗性示例（对抗神经网络的输入示例）**（即**旨在引起分类错误的恶意输入**）。

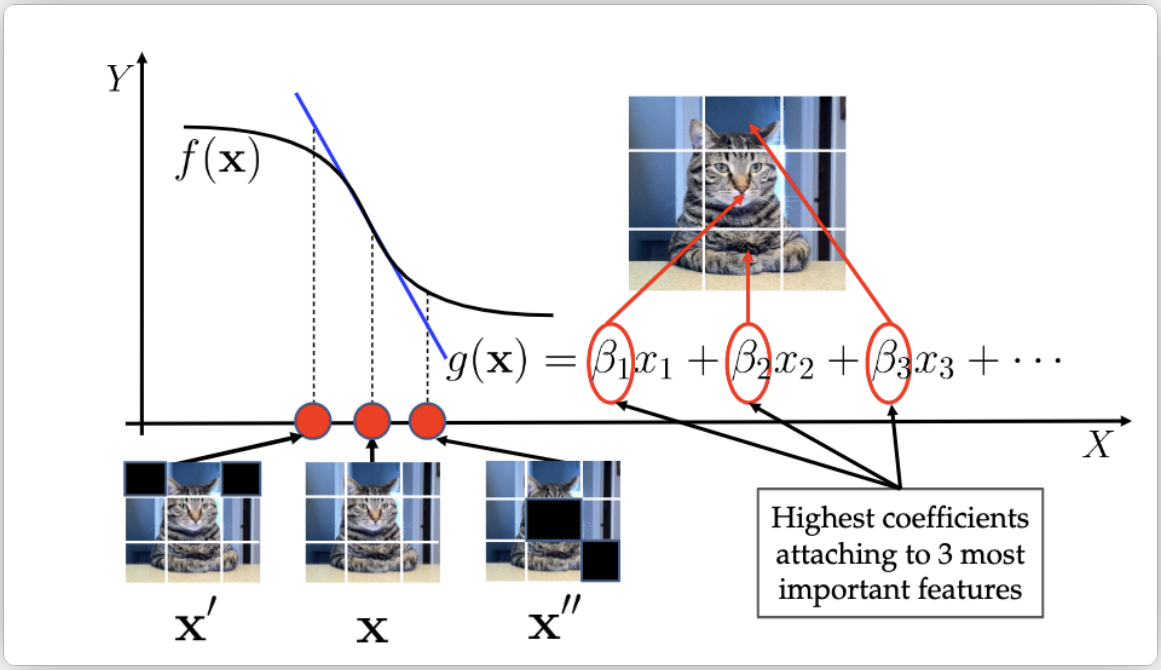
基于向后传播的方法。 基于反向传播的方法利用深度神经网络的梯度来推断特征的重要性。 梯度可以是**分类器输出相对于输入或隐藏层的偏导数。** 通过将输出传播回输入，这些方法可以直接计算输入要素的权重。 对于图像分类器，基本方法是使用相对于图像或视频帧中输入像素的输出梯度来计算特征“显着图（saliency map）”。 以后的工作通过逐层应用显着性图或映射像素组来改进此想法。

基于向后传播的方法面临“零梯度”的挑战。 在神经网络内部，**激活函数通常具有饱和部分，并且相应的梯度将变为零。** 零梯度使得“显着性图”**很难（即使不是不可能）回溯重要特征。** 最近的工作试图通过近似来解决这个问题。 但是，这牺牲了解释的准确性。

Blackbox Explanation Methods

黑盒解释方法不需要理解诸如网络结构和参数等分类器的内部结构。取而代之的是，**他们将分类器视为黑盒子然后通过对于输入输出的观察来进行分析**（即模型推断方法）。这个类别里面最有代表性的工作是 LIME 。给定一个输入 x (比如一个图片)， LIME系统性的扰动x来从x的邻近特征区域里面获取一系列新的人工样本 （比如 Figure 2中的x ′ 和x ′′ ）。我们将这些人工样本放到目标分类器 f(x) 里面去获取相应的样本， 然后我们用线性回归模型 g(x) 来拟合这些数据。这个 g(x) 试图去模拟 f(x)在特征空间中位于输入样本附近区域的决策。LIME假设在输入样本附近的局部决策区间是线性的，因此用线性回归模型来局部代表 f(x) 的决策是合理的。由于线性回归是可自解释的，**因此LIME可以基于回归系数来标定重要的特征。** SHAP 的最新工作试图通过将权重添加到人工生成的数据样本来扩展LIME。 其他工作建议使用其他线性模型（例如决策树和决策集）来逐步逼近目标检测边界。

> 上面所说的扰动生成新样本，再进行拟合训练说的应该是源代码中的`data_inverse`函数。因为是黑盒子，所以通过扰乱输入来获得新的样本新的输入，后面再把这些新的输入放到训练好的RNN中来进行预测得到新的输出，继而借此判断哪部分特征为重要特征?



附带说明一下，我们想澄清一下，机器学习的解释与特征选择方法（例如主成分分析（PCA），稀疏编码或卡方统计）完全不同。 解释方法旨在识别特定输入实例x的关键特征，以具体说明实例x的分类方式。 另一方面，特征选择方法（例如PCA）通常在对整个训练数据进行训练之前应用，以减小特征尺寸（以加快训练速度或减少过度拟合），这无法解释如何做出特定的分类决策。

EXPLAINING SECURITY APPLICATIONS

尽管深度学习显示出构建安全应用程序的巨大潜力，但相应的解释方法却大大落后。 结果，缺乏透明度会降低信任度。 首先，如果安全从业人员不了解如何做出关键决策，他们可能不信任深度学习模型。 其次，如果安全从业人员无法对分类错误（例如，由有偏差的训练数据引入的错误）进行故障排除，则需要担心的是，这些错误可能在以后的实践中被放大。 在下文中，我们介绍了在最近的深度学习上获得成功的两个关键安全应用程序。 然后，我们讨论为什么现有的解释方法不适用于安全应用程序。

Deep Learning in Security Applications

在本文中，我们专注于两类重要的安全应用程序：二进制逆向工程和恶意软件分类。

二进制逆向工程。 深度学习在二进制分析中的应用包括识别功能边界，精确定位函数类型签名和跟踪相似的二进制代码。 更具体地说，有研究者使用双向RNN改进函数边界识别并获得近乎完美的性能；有使用RNN准确跟踪二进制文件中的参数和函数类型的；最近还有研究者运用MLP对控制流程图进行编码以查明易受攻击的代码片段。

恶意软件分类。现有作品主要使用MLP模型进行大规模恶意软件分类。 例如，研究人员已经训练MLP在二进制代码级别检测恶意软件并对Android恶意软件进行分类。 最近，Wang等研究者提出了一种基于审计日志的用于检测恶意软件的抗攻击神经网络。总的来说，对于恶意软件分类，现有的工作主要通过使用MLP模型来进行大规模恶意样本聚类。

一个关键的观察结果是，与CNN相比，**RNN和MLP被这些安全应用程序更广泛地采用。** 原因是RNN被设计为处理顺序数据，**而顺序数据在处理较长的二进制代码序列中表现出色。** 特别是，**双向RNN可以捕获每个十六进制之间的输入序列中的双向依存关系。** 对于恶意软件分类，MLP因其高效而被广泛使用。 另一方面，由于CNN可以利用特征在2D图像上的分组效果，因此在图像上表现良好。 **这些安全应用程序没有这样的"类矩阵"数据结构**，无法从使用CNN中受益。

Why Not Existing Explanation Methods

直接将现有的解释方法应用于安全应用程序存在关键挑战。 在表1中，我们总结了所需的属性，以及为什么现有方法无法提供它们。

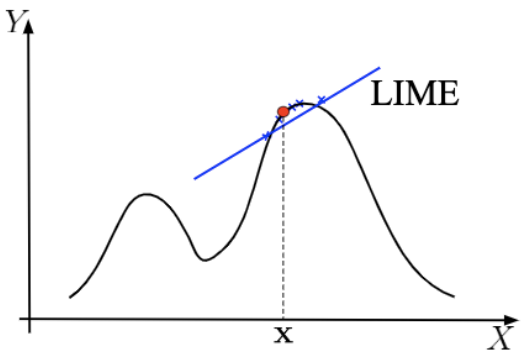
Explanation Method	Support RNN/MLP	Local Non-linear	Support Blackbox	Representative Works
Whitebox method (forward)	◐	○	◐	Occlusion [17, 32, 74, 76], AI^2 [19],
Whitebox method (backword)	◐	○	○	Saliency Map [3, 54, 57], Grad-carm [50], DeepLIFT [53]
Blackbox method	◐	○	●	LIME [45], SHAP [34], Interpretable Decision Set [31]
Our method LEMNA	●	●	●	LEMNA

Table 1: Design space of explainable machine learning for security applications (●=true; ○=false; ◐=partially true).

Supporting RNN And MLP。 上述安全应用程序的模型选择与现有解释方法之间显然不匹配。 现有的大多数解释方法都是为CNN与图像分类器一起使用而设计的。 但是，如上一节所述，我们关注的安全应用程序主要采用RNN或MLP。 由于模型不匹配，现有的解释方法不太适用。 例如，包括"显着性图" 和激活差异传播的反向传播方法需要对CNN的卷积层和池化层进行特殊操作，而RNN或MLP中不存在。

黑盒方法（例如LIME）也不能很好地支持RNN（稍后将通过我们的实验进行验证）。 **像LIME之类的方法假定函数是独立的，但是RNN违反了该假设，后者明确地对顺序数据的依赖性进行建模。**

Supporting Locally Non-linear Decision Boundary。 大多数现有方法（例如LIME）都采用决策边界的局部线性。 但是，当局部决策边界是非线性的时（对于大多数复杂的网络都是如此），那些解释方法会产生严重的错误。 图3a显示了一个示例，其中x周围的决策边界是高度非线性的。 换句话说，线性部分被严格限制在很小的区域。 典型的采样方法很容易击中线性区域以外的人工数据点，从而使线性模型很难近似x附近的决策边界。 在稍后的实验中（第5节），我们确认简单的线性逼近会大大降低解释的逼真度。



(a) Linear regression model. (b) Mixture regression model.

Figure 3: Approximating a locally non-linear decision boundary. The linear regression model (a) can easily make mistakes; Our mixture regression model (b) achieves a more accurate approximation.

Supporting Blackbox Setting。 尽管白盒方法和黑盒方法都有其应用场景，但是对于安全应用程序来说，黑盒方法仍然更为理想。 值得注意的是，人们使用预先训练的模型（例如，Dyninst 中的“双向RNN”，“前缀树”）并不少见，其中详细的网络架构，参数或训练数据都不是全部可用。 即使可以强制某些前向传播方法在黑盒设置下工作（通过放弃对中间层的观察），也不可避免地导致性能下降。

Summary。 在本文中，我们旨在通过为安全应用程序开发专用的解释方法来弥合差距。 我们的方法旨在在黑盒环境下工作，并有效支持流行的深度学习模型，例如RNN，MLP和CNN。 更重要的是，该方法需要实现更高的解释精确度以支持安全性应用程序。

OUR EXPLANATION METHOD

为了实现上述目标，我们设计和开发了LEMNA。 在高等级设计上，我们将目标深度学习分类器视为黑盒，并通过模型逼近来获得解释。 为了提供高精确度的解释，LEMNA需要采用与现有方法截然不同的设计路径。 首先，我们引入fused lasso来处理安全性应用程序(**处理特征间的依赖关系**)和RNN中经常遇到的函数依赖问题（例如，时间序列分析，二进制代码序列分析）。 然后，我们将fused lasso融入到混合回归模型中，以此来拟合局部非线性决策边界来支持复杂的安全应用程序。接下来，我们首先讨论我们设计背后的原理。然后我们将讲述如果将这些设计整合成一个单独的模型，以此来同时处理特征依赖以及局部非线性。最后，我们介绍如何利用LEMNA来得到高精度的解释。

Insights behind Our Designs

Fused Lasso是一种通常用来获取特征依赖的惩罚项，能有效的处理像RNN一类深度学习中的特征依赖。总体而言，Fused Lasso迫使LEMNA将相关/相邻的特征组合起来产生有意义的解释。接下来，我们介绍具体的细节。

```
Lasso 回归：
1. https://blog.csdn.net/pxhdky/article/details/82960659
2. https://www.jiqizhixin.com/graph/technologies/289a87c2-a557-42cd-b2de-6a04d6e258e3

正则化是一种回归的形式，它将系数估计（coefficient estimate）朝零的方向进行约束、调整或缩小。也就是说，正则化可以在学习过程中降低模型复杂度和不稳定程度，从而避免过拟合的危险

而Lasso 回归主要为用于防止出现特征数量过大的情况下出现过拟合的情况，Lasso回归可以让非重点特征的系数参数接近于0而凸显重要特征的系数；从而达到一个筛选特征的目的，将相关/相邻的特征组合起来产生有意义的解释。
```

为了从一组数据样本中学习模型，机器学习算法需要最小化损失函数L(f(x), y)，该函数定义了模型在真实标签和预测标签之间的差异。 例如，要从具有N个样本的数据集中学习线性回归模型f(x) = βx + a，学习算法需要使用最大似然估计（MLE）来针对参数β最小化以下方程式

$$L(f(\mathbf{x}), y) = \sum_{i=1}^{N} \|\boldsymbol{\beta}\mathbf{x}_i - y_i\| . \qquad (1)$$

在这里 Xi 是训练样本，可以表示成一个多维的特征向量 (x1，x2，…，xM)T表示。 xi的标签表示为yi。 向量β=（β1，β2，··βM）包含这个线性模型的系数。 //·//是衡量模型预测与真实标签之间差异的L2范数。fused lasso可以作为一个惩罚项引入学习算法中作为一个损失函数。 以线性回归为例，fused lasso表现为对系数施加的约束，即

$$L(f(\mathbf{x}), y) = \sum_{i=1}^{N} \|\boldsymbol{\beta}\mathbf{x}_i - y_i\| ,$$
$$\text{subject to} \sum_{j=2}^{M} \|\beta_j - \beta_{j-1}\| \le S . \qquad (2)$$

当学习算法使损失函数最小时，Fused Lasso 强制使得相邻特征间的系数之间的差距在一个S范围内。 结果，这个惩罚项会迫使学习算法为相邻特征分配相等的权重。 直观上来说，这个可以被认为是驱使一个学习算法聚合一组特征，然后根据特征群组来解释模型。

安全应用程序，比如时间序列分析和代码序列分析，通常需要使用RNN来对特征之间的依赖性进行建模，由此得到的聚类器依据特征的共存来做出分类决策。如果我们用一个标准的线性回归模型（比如 LIME）来得到一个解释，我们将无法正确的拟合一个局部决策边界。这是因为一个线性回归模型将特征独立对待，无法捕捉到特征依赖。

通过在拟合局部决策边界的过程中引入Fused Lasso, 我们期待得到的线性模型有如下形式：

$$f(\mathbf{x}) = \beta_1 x_1 + \beta_2(x_2 + x_3) + \beta_3(x_4 + x_5) + \cdots + \beta_k x_M\,, \quad (3)$$

在上面的形式中，特征被群组起来。因此，重要的特征有可能被选取成一个或者多个群组。具象的对这个过程进行建模的LEMNA可以推导出精确的解释，尤其是RNN所做出的决策。我们通过Figure 1中的情感分析的例子来解释这个思路。通过引入Fused Lasso, 一个回归模型将一起考虑相邻的特征（比如，一个句子中的相邻单词）。当我们推导解释时，我们的模型不再简单的抓住单词“not”， 同时我们还能精确的抓住短语“not worth the price”来作为情感分析的解释结果。

Mixture Regression Model 使得我们可以精确的拟合局部非线形决策边界。如Figure 3b 所示，一个混合回归模型是多个线性模型的组合。它使得拟合更加有效：

$$y = \sum_{k=1}^{K} \pi_k(\boldsymbol{\beta}_k \mathbf{x} + \epsilon_k)\,, \quad (4)$$

在上面的公式中，K 是一个超参数，代表着混合模型中线性部件的总个数。πk表示的是对应的部件的权重。给定足够的数据，不论一个聚类器有着线性的还是非线性的决策边界，这个混合模型都可以近乎完美地拟合这个决策边界（使用一个有限集合的线性模型）。由此，在深度学习解释的问题中，这个混合回归模型避免了前面提到的非线性问题，从而得到了更精确的解释。

为了更好的阐释这个思路，我们使用Figure 3中的例子。如该图所示，一个标准的线性拟合无法保证输入x 附近的样本仍然在线性局部空间内。这可能轻易导致一个不精确的拟合以及低精度的解释。我们的方法，如 Figure 3b所示，用一个多边的边界来拟合局部决策边界（每一条蓝色的直线代表了一个独立的线性回归模型）。其中最好的拟合是穿过数据点 x 的红线。由此，我们的拟合过程可以产生一个最好的线性回归模型来定位重要的特征。

Model Development

我们把Fused Lasso作为正则项加到mixture regression model中，为了估计模型的参数，我们需要求解如下的优化方程（最小化方程式）：

$$\begin{aligned} &L(f(\mathbf{x}), y) = \sum_{i=1}^{N} \|f(\mathbf{x}_i) - y_i\|\,, \\ &\text{subject to} \sum_{j=2}^{M} \|\beta_{kj} - \beta_{k(j-1)}\| \le S\,,\ k = 1, \ldots, K\,. \end{aligned} \quad (5)$$

其中f是regression mixture model，β是参数。与标准线性回归不同，我们的优化目标很棘手，我们不能简单地利用MLE进行最小化。 为了有效地估计混合回归模型的参数，我们采用了另一种方法（期望最大算法（E-M））。 首先，我们以概率分布的形式表示混合回归模型（使用EM算法需要先观察数据满足什么样的分布？）：

$$y_i \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{\beta}_k \mathbf{x}_i, \sigma_k^2) . \qquad (6)$$

其中π，β，σ2是需要估计的参数。首先我们随机初始化参数，然后重复进行EM算法的E步和M步直到算法收敛。下面我们简单介绍一下具体算法。

EM（Expectation Maximization）期望最大化算法 ：
1. 含推导过程：https://zhuanlan.zhihu.com/p/36331115
2. http://www.csuldw.com/2015/12/02/2015-12-02-EM-algorithms/
3. https://zhuanlan.zhihu.com/p/40991784

EM算法的基本思想：存在隐含参数与分布参数相互依赖互相影响。根据分布参数来确定调整隐含参数为E步，根据隐含参数确定调整分布参数为M步，如此往复，直到参数基本不再发生变化或满足结束条件为止。

文章1中的解析：EM 算法解决这个的思路是使用启发式的迭代方法，既然我们无法直接求出模型分布参数，那么我们可以先猜想隐含参数（EM 算法的 E 步），接着基于观察数据和猜测的隐含参数一起来极大化对数似然，求解我们的模型参数（EM算法的M步）。由于我们之前的隐含参数是猜测的，所以此时得到的模型参数一般还不是我们想要的结果。我们基于当前得到的模型参数，继续猜测隐含参数（EM算法的 E 步），然后继续极大化对数似然，求解我们的模型参数（EM算法的M步）。以此类推，不断的迭代下去，直到模型分布参数基本无变化，算法收敛，找到合适的模型参数。

另一种对于EM算法的理解：EM算法通过引入隐含变量，使用MLE（极大似然估计）进行迭代求解参数。通常引入隐含变量后会有两个参数，EM算法首先会固定其中的第一个参数，然后使用MLE计算第二个变量值；接着通过固定第二个变量，再使用MLE估测第一个变量值，依次迭代，直至收敛到局部最优解。

一个最直观了解 EM 算法思路的是 K-Means 算法。在 K-Means 聚类时，每个聚类簇的质心是隐含数据。我们会假设 K 个初始化质心，即 EM 算法的 E 步；然后计算得到每个样本最近的质心，并把样本聚类到最近的这个质心，即 EM 算法的 M 步。重复这个 E 步和 M 步，直到质心不再变化为止，这样就完成了 K-Means 聚类。

理解一时爽，推导火葬场～

从公式中可以看出，y服从一个由K个高斯分布（正态分布）组成的distribution。每个Gaussian（高斯）有自己的平均值（β）和方差 （σ2）。在每一次迭代中首先进行E步，我们把每个样本点分配到一个Gaussian。这里，我们使用的分配方法就是标准的EM算法的E步。完成E步后，根据新的数据分配结果，我们使用每一个Gaussian自己的数据更新它的mean和variance。更新variance的方法

和标准EM相同，但是因为我们加了Fused Lasso的正则项在mean上，所以更新mean相当于求解如下优化方程：

$$L(x,y) = \sum_{i=1}^{N_k} \|\boldsymbol{\beta}_k \mathbf{x}_i - y_i\|,$$
$$\text{subject to} \sum_{j=2}^{M} \|\beta_{kj} - \beta_{k(j-1)}\| \leq S, \tag{7}$$

我们重复E步和M步直到模型收敛，进而输出模型参数。

这里是借助了EM算法来求解fused lasso 的约束函数从而得到更加精准的特征集合。

Applying the Model for Explanation

通过增强的混合回归模型，我们现在介绍如何使用本文提出的模型解释神经网络的结果。具体来说解释过程分为以下两步：近似局部的决策边际和生成解释。

近似局部的决策边际。给定一个输入样本，生成解释的关键是近似深度学习模型的局部决策边际，从而获知聚类该样本的重要特征。为了实现这个目的，我们首先使用 **[45]** 中的方法生成一组人工样本。然后我们使用这些数据来模拟目标模型的局部决策边际。有两个可能的方法来实现模拟。第一种是使用一个混合回归模型进行多类分类。第二个是对每一个类使用一个混合回归模型。在本文中考虑到计算复杂度，我们使用第二种方法。

Deriving Explanations。 如前所述，对于一个给定的样本点，我们的解释是抓取目标模型聚类该样本时依据的重要特征。首先，我们通过上述的方法得到一个混合线性回归模型（mixture component）。这个线性回归的参数可以被视为特征的重要性。具体来说，我们把拥有大系数的特征作为重要的特征，同时选择最重要的一小组特征作为解释。

需要注意的是， 虽然LEMNA是为了非线性模型和特征相关性设计的，这不意味着LEMNA不能解释其他的深度学习模型（MLP和CNN）。事实上，LEMNA是可以根据所解释的深度学习模型调整的。比如，通过增加fused lasso的超参数S，我们可以放松这个正则项，进而使LEMNA适用于假设特征独立的深度学习模型。

EVALUATION

在本节中，我们评估我们的解释方法在两个安全应用程序上的有效性：恶意软件分类和二进制逆向工程。 本部分通过一系列精确度指标来重点评估解释的准确性。 在下一部分（第6节）中，我们将介绍LEMNA的实际用例，以了解分类器行为，排除分类错误并修补分类器错误。

Experimental Setup

我们将LEMNA应用于两个安全应用程序：使用RNN检测用于反向工程二进制代码的“函数开始位置”，以及基于MLP对PDF恶意软件进行分类。 下面，我们介绍有关这两个安全应用程序，LEMNA的实现以及基本的详细信息。

二进制逆向工程。 将二进制代码转换为汇编代码的二进制代码逆向工程是：
（1）检查和检测恶意软件
（2）加强软件安全性
（3）生成安全补丁
多年来，二进制分析主要是由经验丰富的安全分析师手动完成的。 最近，研究人员表明，训练有素的RNN可以帮助处理关键的逆向工程步骤，例如发现函数开始位置，这可以大大节省人工。 考虑到检测函数开始位置的重要性（即所有二进制代码逆向工程都需要知道函数的开始位置），我们选择此应用程序来测试LEMNA。

接下来，我们按照 **[52]** 中的方法使用了2200个程序来训练RNN网络。 我们在x86架构和gcc编译器下分别使用四个不同的优化级别O0，O1，O2和O3来编译这些二进制文件。 这将产生4个训练数据集，每个优化级别一个。

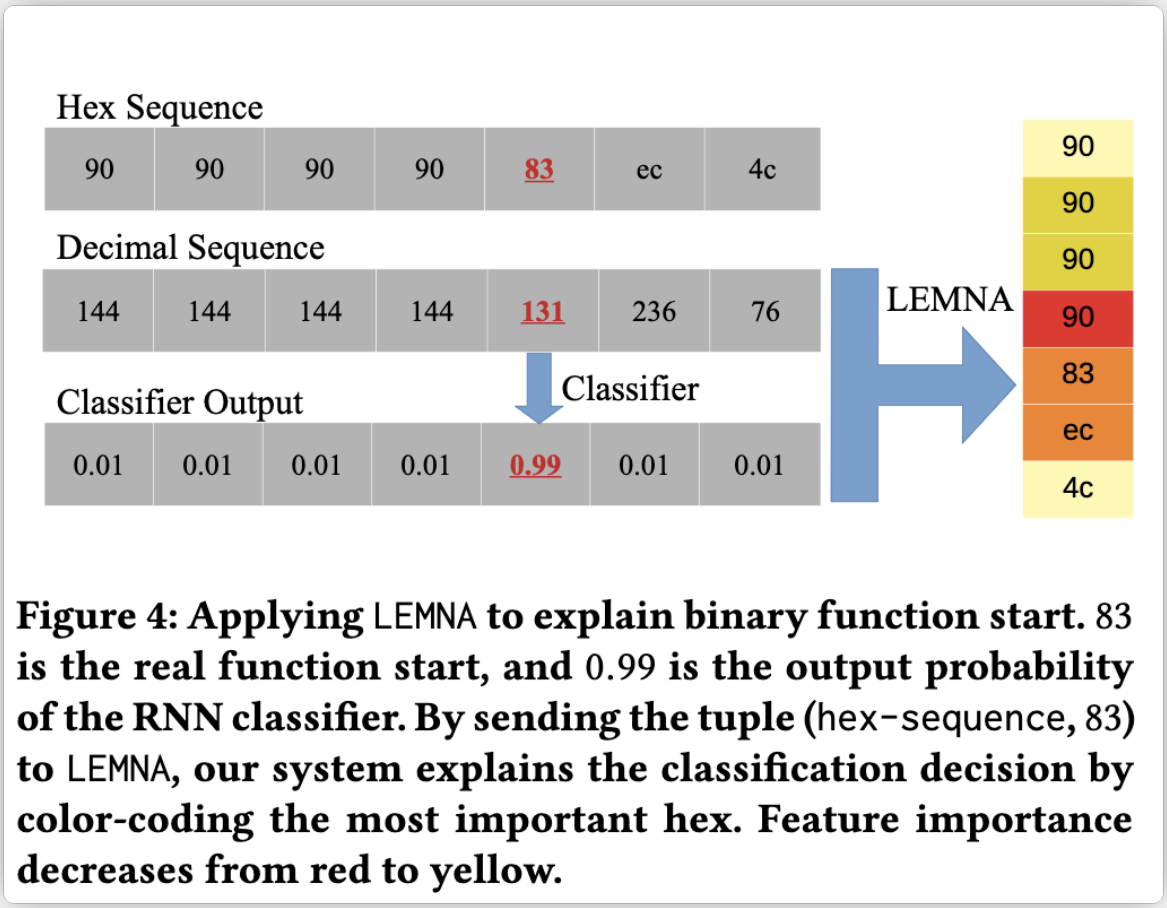


Figure 4: Applying LEMNA to explain binary function start. 83 is the real function start, and 0.99 is the output probability of the RNN classifier. By sending the tuple (hex-sequence, 83) to LEMNA, our system explains the classification decision by color-coding the most important hex. Feature importance decreases from red to yellow.

数据集中的每个二进制都以十六进制代码序列的形式呈现。 如图4所示，我们首先将十六进制代码转换为其十进制值，然后将序列中的每个元素都视为一个特征。 为了进行训练，序列中的每个元素都带有“a function start”或“ not a function start”的标签。 如图4所示，假设原始二进制代码为“ 90 90 90 83 ec 4c”，并且函数起始位置为“ 83”，则标签向量为（0，0，0，0，1，0，0）。 接下来，我们同样依据 **[52]** 将长序列切分成最大长度为200的短序列。然后，将序列馈入RNN。 我们使

用Keras 训练模型，并以Theano作为后端。 我们使用70％的样本进行训练，其余30％进行测试，以随机方式拆分数据集。

如表2所示，检测精度极高，在所有情况下的准确度均达到98.57％或更高。 结果与[52]中报道的结果相当。 RNN的超参数可以在附录C中找到。

Application	Binary Function Start				PDF Malware
	00	01	02	03	
Precision	99.99%	99.65%	98.57%	99.53%	99.12%
Recall	99.97%	99.49%	98.81%	99.06%	98.13%
Accuracy	99.99%	99.99%	99.99%	99.99%	98.64%

Table 2: Classification accuracy of the trained classifiers.

PDF Malware Classifier。我们遵循[21，48]基于广泛使用的数据集（4999个恶意PDF文件和5000个良性文件）构建基于MLP的恶意软件分类器[55]。 我们遵循[55，58]为每个文件提取135个特征。这些功能是由研究人员根据元数据和PDF的结构（例如对象标记的数量和javascript标记的数量）手动制作的。 完整的功能列表可在Mimicus [1]中找到。 我们遵循标准方法将特征值转换为二进制表示形式[41]（即非零特征值转换为1），这有助于避免某些高价值特征使训练过程发生偏差。 像以前一样，我们随机选择70％的数据集（恶意软件和良性软件 1：1）作为训练数据，其余30％作为测试数据。 如表2所示，我们的准确性和召回率均高于98.13％，与[55]相似

LEMNA Implementation。我们将上面提到的RNN作为LEMNA的目标模型。给定一个输入样本，LEMNA模拟这个目标聚类器然后解释聚类结果。这里，“解释”代表着一个输入样本中最重要的特征。对于“函数开始位置”聚类器，我们在Figure 4中展示了一个例子。给定一个样本数值序列以及RNN标定的函数开始位置（即“83”），LEMNA标定序列中拥有最大贡献的字节集合。在这个例子中，**“83”是函数开始字节，而且LEMNA将函数开始位置前的“90”标记为RNN聚类的最重要的原因**。

LEMNA有三个可配置的超参数。首先为了拟合局部决策边界，我们构建N个数据样本来做模型拟合。第二个以及第三个参数分别是混合模块中的K以及Fused Lasso中的阈值S。在函数开始位置检测中，我们将这些参数分别设置为：N=500, K=6, S=1e – 4。对于恶意软件分类，我们将参数设置为：N = 500，K = 6，S = 1e4。 请注意，参数S的设置非常不同，**因为恶意软件分析功能相对独立，而二进制分析函数具有较高的依赖级别**。 我们修复了这些参数以运行我们的大部分实验。 稍后，我们有专门的部分对参数设置执行灵敏度测试（这表明LEMNA对这些超参数不敏感）。

LEMNA的计算成本。 **LEMNA的计算成本相对较低**。 对于这两个安全应用程序，为给定实例生成说明的时间约为10秒。 该计算任务进一步受益于并行化。 例如，使用配备Intel Xeon CPU E5-2630的服务器，一台Nvidia Tesla K40c GPU和256G RAM，大约需要2.5个小时来解释30个线程对O0的所有25，040个二进制测试序列。

Comparison Baselines. 我们使用两个基线进行比较。 首先，我们使用最新的黑盒方法LIME 作为比较基准。 LIME 已被用来解释图像分类器和NLP应用。 在安全应用程序和RNN上的性能尚不清楚。 为了公平地比较，我们还将LIME配置为N = 500，这是用于拟合线性回归模型的人工样本的数量。 其次，我们使用随机特征选择方法作为基准。 给定输入后，Random方法将随机选择特征作为分类结果的解释。

Fidelity Evaluation

为了验证解释的正确性（精确度），我们进行了两个阶段的实验。 在第一阶段，我们直接检查相对于原始决策边界的局部逼近的准确性，这很可能会给出解释准确性的初步估计。 在第二阶段，我们对解释精度进行端到端评估。 我们设计了三个保真度测试，以显示所选特征是否确实是分类结果的主要贡献者。

评估1：局部近似精度。 通过比较近似决策边界和原始决策边界，可以直接计算该度量。 我们测量均方根误差Root Mean Square Error（RMSE）：

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - \hat{p}_i)}{n}},$$

其中pi表示从目标深度学习分类器获得的单个预测，ˆpi表示从解释方法获得的近似预测，并且 n 表示测试数据样本的总数。 更具体地说，我们从给定的分类器和一组测试数据样本开始。对于每个测试数据样本xi，我们首先使用分类器获得预测概率。 然后对于xi，我们根据等式（6）生成回归模型，该模型可以产生估计的预测概率ˆpi。 对所有样本进行这些步骤后，我们获得了预测向量P =（p1，p2，...，pn）和相应的近似向量ˆP =（ˆp1，ˆp2，...，ˆpn）。 最后，我们基于两个向量对RMSE进行计算机处理。 较低的RMSE意味着近似的决策边界（ˆP）接近真实边界（P），表明解释的精确度更高。

(a) Input Image. (b) Explanation. (c) Deduc. test. (d) Augme. test. (e) Synthet. Test.

Figure 5: We use an image classifier as an toy example to explain the fidelity test. Figure 5a is the original input image ("sweater"). Figure 5b is the explanation produced by LEMNA where important features (pixels) are highlighted in red. Figure 5c–5e are three testing instances we generated to test the fidelity of the explanation.

评估2：端到端保真度测试。 为了验证所选功能的正确性，我们设计了三个端到端保真度测试。 为了帮助读者理解测试过程，我们使用"图像分类器"为例5。 对于其他分类器，该过程的工作方式相同。 如图5所示，图像分类器经过训练，可以将"鞋子"从"毛衣"中进行分类。 图5a是标签为"sweater"的输入图像（x）。 在图5b中，解释方法通过用红色突出显示重要像素（特征）来解释分类的原因。 我们将所选要素表示为Fx。 为了检验说明的真实性，我们有以下三种想法：

- 如果正确选择了特征Fx，则从输入x中移除Fx将导致将该图像分类到其他标签，即“鞋子”（图5c）
- 如果正确选择了特征Fx，则将Fx的特征值添加到“鞋子”的图像中可能会导致分类错误，即将其分类为“毛衣”（图5d）。
- 如果正确选择了特征Fx，我们可以制作仅包含Fx中特征的合成图像，并且该合成图像很可能被归类为“毛衣”（图5e）。

利用这些直觉，我们构造了3种不同的保真度测试来验证所选功能。 更正式地说，给定输入实例x及其分类标签y，LEMNA会将一小部分重要特征(Fx)标识为“解释”。 然后，我们按照以下步骤生成3个测试样本t(x)1，t(x)2和t(x)3用于特征验证：

- 特征推导测试：我们通过从实例x消除所选特征Fx来构造样本t(x)1。
- 功能增强测试：我们首先从相反的类别中选择一个随机实例r（即，只要r的标签不是y）。 然后我们通过用Fx替换实例r的特征值来构造t(x)2。
- 综合测试：我们将t(x)3构造为综合实例。 我们保留所选要素Fx的要素值，同时为其余要素随机分配值.

此实验中的关键变量是被选作“解释”的重要特征的数量（即|Fx|）。 直观上，较大的|Fx| 可能会产生更好的解释精准度，但会损害结果的可解释性。 我们要保持|Fx| 很小，以便人类分析人员能够理解。 对于每个分类器，我们在测试数据集（全部数据的30％）上运行精确度测试。 给定测试数据集中的一个实例x，我们生成3个样本，每个精确度测试一个。 我们将这3个样本送入分类器，并检查positive classification rate (PCR)。 PCR可测量仍归为x原始标签的样品所占的比例。 请注意，此处的“positive”并不意味着“恶意软件”或“函数开始位置”。 这只是意味着**新样本仍被归类为x的原始标签。 如果特征选择正确，我们期望特征推导样本返回低PCR，特征增强样本返回高PCR，而合成测试样本返回高PCR。**

Experimental Results

我们的实验表明，在所有保真度指标上，LEMNA均优于LIME和随机基线。

局部近似精度。 如表3所示，LEMNA的RMSE比LIME小。 这个观察对于恶意软件分类器和函数开始位置检测均适用。 LIME的最佳结果的RMSE为0.1532，仍比LEMNA的较差结果（0.0196）高出近10倍。 这个结果证实了我们的混合回归模型能够比简单的线性模型建立更精确的近似值。 请注意，此指标不适用于随机基线，因为随机基线不会构成决策边界。

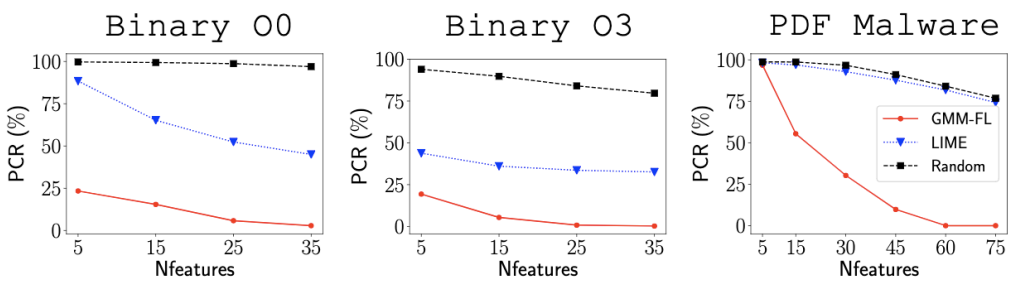
Method	Binary Function Start				PDF malware
	00	01	02	03	
LIME	0.1784	0.1532	0.1527	0.1750	0.1178
LEMNA	0.0102	0.0196	0.0113	0.0110	0.0264

Table 3: The Root Mean Square Error (RMSE) of local approximation. LEMNA is more accurate than LIME.

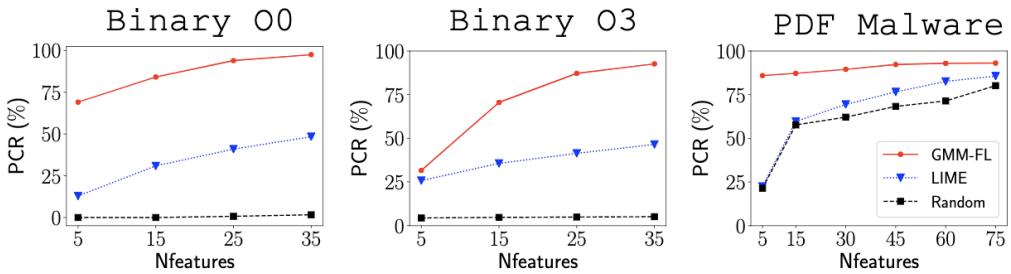
精确度测试。 图6a显示了特征移除实验的结果。通过移除LEMNA标定的最重要的5个特征，函数开始位置的PCR降低到了25％甚至更低。 考虑到分类器的极高准确性（99.5％+，请参阅表2），这个剧烈的降低标志着这个小的特征集合对于聚类结果是及其重要的。

Figure 6b 展示了特征加强的结果。这个结果跟之前的测试是一致的：（1）通过加入一个小集合的特征，我们可以实现聚类结果的反转；（2）我们的方法显著的优于其他的两种方法。

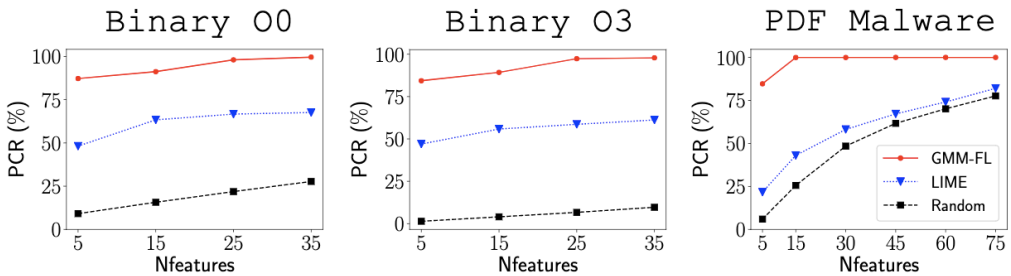
Figure 6c 展示了在特征生成实验中，LEMNA有同样的优势。利用LEMNA挑选的重要特征，这些生成的样本极有可能会被归类成原始的标签。使用前五个最重要的特征，这些生成的样本有85%-90%的概率会被归类成原来的标签，标志着核心的特征已经被标定。



(a) Feature Deduction test. A lower PCR reflects a higher explanation fidelity.



(b) Feature Augmentation test. A higher PCR reflects a higher explanation fidelity.



(c) Synthetic test. A higher PCR reflects a higher explanation fidelity.

Figure 6: Fidelity test results. y-axis denotes the positive classification rate PCR and y-axis denote the number of selected features NFeature by the explanation method. Due to the space limit, the results of Binary-O1 and O2 are shown in Appendix-D.

在所有这三个测试中，我们的LEMNA都大大优于LIME和随机基线。 有趣的是，对于恶意软件分类器，LIME的性能与随机特征选择一样差。 这是因为特征向量稀疏，这会损害决策边界的“平滑度”。 LIME很难准确地估计不光滑的边界，这再次验证了我们的设计想法。 考虑到与图像分析任务相比，安全性应用程序需要更高的解释精度，因此我们的系统更适合于安全性应用程序。

超参数的灵敏度。 最后，我们测试如果参数设置不同，结果将如何变化。 我们测试了大量参数配置，发现我们的结论保持一致。 由于篇幅所限，我们在表4中汇总了主要结果。这三个超参数是用于模型拟合的“制作数据样本的数量”（N），“混合模型系数的总数”（K）和“ fused lasso的阈值”（S）。 表4列出了O0数据集上的二进制函数启动检测器的结果。 我们显示了4组配置，其中我们一次更改一个参数。 对于精确度测试，我们将选定特征的数目固定为25以计算PCR。 结果证实，更改超参数不会显着影响LEMNA的性能。

(N, K, S)	RMSE	Deduc. test	Augme. test	Synthet. test
(500, 6, 1e-4)	0.0102	5.79%	93.94%	98.04%
(300, 6, 1e-4)	0.0118	5.94%	94.32%	98.18%
(500, 4, 1e-4)	0.0105	5.80%	93.71%	97.89%
(500, 6, 1e-3)	0.0114	5.83%	93.21%	97.73%

Table 4: Hyper-parameters sensitivity testing results.

APPLICATIONS OF ML EXPLANATION

到目前为止，我们已经验证了解释结果的真实性。 在本节中，我们介绍LEMNA的实际应用。 我们通过case study来展示如何用我们的解释结果来帮助分类器建立可信度以及定位分类错误的原因：1）建立对训练有素的分类器的信任； 2）解决分类错误； 3）并系统地修补目标错误。 在下文中，我们主要关注于二进制逆向工程应用程序，因为深度学习的这一应用领域是相对较新的，并且未被很好地理解。 我们有对PDF恶意软件分类器执行了相同的分析，结果在附录E中。

Understanding Classifier Behavior

我们解释方法的主要应用是评估聚类器的可靠程度并且建立可信度。我们认为聚类器的可靠度和可信度并不一定来自于高的聚类精度。相反的，可信度更可能通过理解模型行为来建立。这里，我们从两个关键的角度来观测来评估聚类器的行为：

（1）抓取常见的原则

（2）发现新的知识。

Capturing Well-known Heuristics (C.W.H.)。 可靠的分类器至少应捕获相应应用程序领域中的众所周知的启发式方法。 例如，在二进制逆向工程领域，安全从业人员积累了一组有用的启发式方法来识别功能开始，其中一些甚至被视为“黄金法则”。 某些“黄金法则”源自应用程序二进制接口（ABI）标准的规范。 例如，如果该函数维护新的帧指针，则ABI要求该函数在开始时存储旧的帧指针（ebp）。 这导致了最常见的序言[push ebp; mov ebp，esp]。 另一套完善的规则来自主流编译器。 例如，GNU GCC经常在函数开始之前插入nop指令，从而使函数对齐以进行架构优化。

Cases	ID	Opt.-level	F. Start	Explanation	Assembling code
C.W.H.	1	00	*55*	5b 5d c3 **55** 89 e5	pop ebx; pop ebp; ret; **push ebp**; mov ebp, esp
	2	01	*53*	5b 90 c3 **53** 83 ec 18	pop ebx; nop; ret; **push ebx**; sub esp,0x18
	3	02	*89*	8d b4 26 00 00 00 00 **89** c1 8b 40 0c	lea esi, [esi+eiz*1+0]; **mov ecx, eax**
	4	03	*56*	90 90 90 90 **56** 53	nop; nop; nop; nop; **push esi**; push ebx
D.N.K.	5	00	*31*	e9 00 f9 ff ff **31** ed 5e	jmp 0xfffff900; **xor ebp, ebp**; pop esi
	6	01	*b8*	90 90 90 **b8** e7 20 19 08 2d e4 20 19 08	nop; nop; nop;**mov eax, 0x81920e7**;sub eax, 0x81920e4
	7	02	*83*	83 c4 1c c3 **83** ec 1c	add esp, 0x1c; ret; **sub esp, 0x1c**
	8	03	*8b*	90 90 90 90 **8b** 44 24 04	nop; nop; nop; nop; **mov eax, DWORD PTR [esp+0x4]**
	9	03	*55*	8d bc 27 00 00 00 00 **55** 57 56	lea edi, [edi+eiz+0x0]; **push ebp**; push edi; push esi
R.F.N.	10	00	*31**	e9 **50 fd** ff ff **31** ed 5e	jmp 0xfffffd50; **xor ebp, ebp**; pop esi
	11	02	*89**	e9 85 fe ff ff 90 **89** c2 31 c0	jmp 0xfffffe8a; nop; **mov edx, eax**; xor eax, eax
	12	03	*a1**	8d b4 26 00 00 00 00 **a1** d0 14 20 08	lea esi, [esi+eiz*1+0]; **mov eax, ds:0x82014d0**
R.F.P.	13	01	*83*	0f b6 c0 c3 **83** ec 1c	movzx eax,al; ret; **sub esp, 0x1c**
	14	02	*b8*	8d 74 26 00 **b8** 01 00 00 00	lea esi, [esi+eiz*1+0x0]; **mov eax, 0x1**
	15	03	*83*	8d 74 26 00 **83** ec 1c c7 04	lea esi, [esi+eiz*1+0x0]; **sub esp, 0x1c**

Table 5: Case study for the binary analysis (15 cases). Our explanation method ranks features and marks the most important features as red, followed by orange, gold, yellow. We also translate the hex code to assembling code for the ease of understanding. Note that the `F. start` refers to the function start detected by the deep learning classifier. The function start is also marked by a black square in the hex sequence. *For false negatives under R.F.N., we present the *real* function start that the classifier failed to detect, and explain why the function start is missed.

通过分析解释结果，我们观察到有力的证据表明深度学习分类器已成功捕获了著名的启发式方法。在表5中，我们显示了4个最具代表性的情况，每个分类器（或优化级别）一个。 在案例1中，分类器正确地检测到函数从“ 55”开始。 然后我们的LEMNA通过突出显示功能的重要性（即附近的十六进制代码）来说明为什么将55标记为功能。 结果符合众所周知的黄金法则，即[push ebp; mov ebp，esp]。 这表明分类器正在以合理的方式做出决策。 同样，Case-2在“ c3”之后捕获函数开始“ 53”。 这对应于编译器引入的一种流行的启发式方法，因为编译器通常通过“ ret”指令（尤其是在O0和O1级别）最终使函数退出。

在情况4中，“ 83”是函数开始位置，LEMNA用红色突出显示“ 90”。 这表明分类器遵循“函数开始之前立即执行nop”规则，这是由编译器在对齐函数之前填充“ nop”引起的。 同样，在案例3中，LEMNA突出显示了填充指令[lea esi，[esi + eiz * 1 + 0]]，这是编译器引入的另一种模式。 总体而言，LEMNA表明，分类器已成功捕获了著名的启发式算法。

在我们的分析过程中，我们观察到众所周知的启发式方法在较低的优化级别（O0，O1）下可广泛应用，但在较高的级别（O2，O3）上覆盖的二进制数并不多。 例如，O0级别的功能的95％以[55 89 E5]开头，与Case-1的启发式匹配。 O1优化的功能中有74％已退出作为结束指令（情况2）。 相反，在O2或O3级别上只有30％的二进制函数与众所周知的启发式匹配，例如，函数末尾的填充指令（“ [90 90 90 90]，” [8d b4 26 00 00 00 00]”。这是直观的感觉，因为更高级别的优化将极大地分散代码结构，从而使黄金规则的效率降低。

Discovering New Knowledge (D.N.K.)。 除了匹配知名的启发式算法，我们还检查了分类器是否选择了在现有知识之外建立新的启发式方法。 对于安全应用程序，我们认为新的启发式方法必须由领域专家来解释。 在二元分析领域，许多潜在有用的启发式方法特定于单个功能，因此很难手动总结所有这些功能。 例如，链接器插入的实用程序函数通常具有唯一的开始代码段，而这些段很少出现在其他位置（例如，_start函数始终以[xor ebp，ebp; pop esi]开头）。 手动组织此类规则不切实际。 但是，这些规则一旦由LEMNA得出，对于领域专家来说将具有直观的意义。

如表5所示，我们分析了解释结果，发现分类器确实学习了新知识。 我们选择五个代表性案例（ID 5–9）。 情况5显示，由于随后的[ed 5e]，检测到“ 31”作为功能启动。 “ [31 ed 5e]”对应于实用程序函数_start的开始（即[xor ebp，ebp; pop esi]）。 这说明我们的解释方法可以帮助总结与特殊功能有关的独特序言。 注意，功能开始“ 31”本身不一定是重要的指标。 实际上，“ 31”表示通常出现在功能中间的操作码（异或）。 是“ [ed 5e]”导致正确的检测。

情况6说明了另一种有趣的模式，其中**“ 2d”(这里是2b还是2d？)**是检测函数从“ b8”开始的最重要特征。 “ 2d”位于模式[mov eax，CONS1; [sub eax，CONS2]，其中CONS1和CONS2是常数，而CONS1-CONS2 = 0或3。此模式仅出现在“ register_tm_clones”和“ deregister_tm_clones”的序言中，它们是事务性存储器的实用函数。 同样，这是检测功能启动的特定功能模式。 Case-7，Case-8和Case-9在功能启动时都具有某些类型的“准备”。

Cases	ID	Opt.-level	F. Start	Explanation	Assembling code
C.W.H.	1	00	*55*	5b 5d c3 **55** 89 e5	`pop ebx; pop ebp; ret;` **push ebp**`; mov ebp, esp`
	2	01	*53*	5b 90 c3 **53** 83 ec 18	`pop ebx; nop; ret;` **push ebx;** `sub esp,0x18`
	3	02	*89*	8d b4 26 00 00 00 00 **89** c1 8b 40 0c	`lea esi, [esi+eiz*1+0];` **mov ecx, eax**
	4	03	*56*	90 90 90 90 **56** 53	`nop; nop; nop; nop;` **push esi**`; push ebx`
D.N.K.	5	00	*31*	e9 00 f9 ff ff **31** ed 5e	`jmp 0xfffff900;` **xor ebp, ebp**`; pop esi`
	6	01	*b8*	90 90 90 **b8** e7 20 19 08 2d e4 20 19 08	`nop; nop; nop;`**mov eax, 0x81920e7**`;sub eax, 0x81920e4`
	7	02	*83*	83 c4 1c c3 **83** ec 1c	`add esp, 0x1c; ret;` **sub esp, 0x1c**
	8	03	*8b*	90 90 90 90 **8b** 44 24 04	`nop; nop; nop; nop;` **mov eax, DWORD PTR [esp+0x4]**
	9	03	*55*	8d bc 27 00 00 00 00 **55** 57 56	`lea edi, [edi+eiz+0x0];` **push ebp**`; push edi; push esi`
R.F.N.	10	00	*31**	e9 **50 fd** ff ff **31** ed 5e	`jmp 0xfffffd50;` **xor ebp, ebp**`; pop esi`
	11	02	*89**	e9 85 fe ff ff 90 **89** c2 31 c0	`jmp 0xfffffe8a; nop;` **mov edx, eax**`; xor eax, eax`
	12	03	*a1**	8d b4 26 00 00 00 00 **a1** d0 14 20 08	`lea esi, [esi+eiz*1+0];` **mov eax, ds:0x82014d0**
R.F.P.	13	01	*83*	0f b6 c0 **c3** **83** ec 1c	`movzx eax,al; ret;` **sub esp, 0x1c**
	14	02	*b8*	8d 74 26 00 **b8** 01 00 00 00	`lea esi, [esi+eiz*1+0x0];` **mov eax, 0x1**
	15	03	*83*	8d 74 26 00 **83** ec 1c c7 04	`lea esi, [esi+eiz*1+0x0];` **sub esp, 0x1c**

Table 5: Case study for the binary analysis (15 cases). Our explanation method ranks features and marks the most important features as red, followed by orange, gold, yellow. We also translate the hex code to assembling code for the ease of understanding. Note that the `F. start` refers to the function start detected by the deep learning classifier. The function start is also marked by a black square in the hex sequence. *For false negatives under R.F.N., we present the *real* function start that the classifier failed to detect, and explain why the function start is missed.

在情况7中，“ [83，ec]”被标记为最重要的特征，与指令[sub esp，0x1c]相对应。 这种形式的指令经常在函数启动时用于准备堆栈框架。 对于情况8，[mov eax，DWORD PTR [esp + 0x4]]被标记为最具指示性的功能。 通常插入此指令以获取函数的第一个参数。 请注意，“ 04”具有红色，这是因为“ 04”被用作[esp + 0x4]的偏移量以获取函数的参数。 如果该偏移量具有不同的值，则该指令不一定是功能启动的指示。 对于情况9，它首先保留以后修改的寄存器（[push ebp; push edi; push esi]）。 调用约定（通用的ABI标准）要求保留这些寄存器，这些函数通常也出现在函数启动时。

总体而言，LEMNA验证了分类器的决策在很大程度上遵循了可解释的逻辑，这有助于建立对这些分类器的信任。

Troubleshooting Classification Errors（解决分类错误）

虽然深度神经网络高度精确，但是他们仍然可能产生错误。这些不应该被忽略，因为他们通常意味着训练不足并且这些不足可能在现实中被放大。我们的解释方法尝试寻找到错误聚类的原因。通过找到这些原因，我们希望能够启发解决这些错误的可行性方案。

Reasons for False Negatives (R.F.N.)。 对于二元分析应用程序，分类器有时会错过实际函数的开始。 如表5所示（在“ R.F.N.”下），给定错误的分类，我们解释“why the real function start is not classified as a function start”。 具体来说，我们将元组（Code-sequence, Real-function-start）输入到LEMNA中，红色特征是无法识别函数启动的原因。 例如，在案例10中，“ [50 fd]”被标记为主要原因，与“ [jmp 0xfffffd50]”相对应。 这条指令几乎都出现在函数中间位置，导致聚类器认为后面的31不是函数开始位置。 这是一个异常情况，因为此“ [50 fd]”恰好是特殊区域.plt的最后一条指令，后跟_start函数。 由于说明“ [mov edx，eax]”和“ [mov eax，ds：0x82014d0]”（通常出现在函数中间），情况11和情况12被错误分类。

Reasons for False Positives (R.F.P.)。 表5还显示了示例，其中分类器选择了错误的函数开始。 在这里，我们将元组（Code-sequence, Real-function-start）输入到LEMNA中，以解释为什么选择了错误的函数开始。 例如，案例13用红色突出显示了代表“ ret”指令的“ c3”。 通常，“ ret”位于函数的末尾以退出，这使下一个字节“ 83”成为函数开始的强力候选者。 但是，Case-13是特殊的，因为出于优化目的，“ ret”实际上位于函数的中间。 Case-14和Case-15都被填充指令[lea esi，[esi + eiz * 1 + 0x0]]所误导，该指令通常用于对齐函数。 但是，在两种情况下，该填充指令实际上都用于在函数内部对齐基本块。

总体而言，LEMNA表明，错误主要是由误导性模式主导实际指标这一事实造成的。 为了减轻此类错误，我们需要在特征空间中查明相应区域并消除误导性图案。

Targeted Patching of ML Classifiers

基于以上结果，我们现在开发自动程序，将以上的一些“想法”转换为修补分类器的修补操作。 为了修补特定的分类错误，我们的想法是识别未充分训练的分类器对应部分。 然后，我们制作有针对性的训练样本以扩大原始训练数据。 具体来说，给定一个错误分类的实例，我们应用LEMNA来找出导致错误的少量特征（Fx）。 通常，这种情况是训练数据中的异常值，并且没有足够的“反例”。 为此，我们的策略是通过添加相关的“反例”（通过使用随机值替换Fx的特征值）来增加训练数据。

我们使用一个示例（表5中的案例10）来描述修补过程。 分类器由于“ [50 fd]”而错过了功能开始，“ [50 fd]”是一个经常出现在功能中间的十六进制模式。 理想情况下，分类器应该选择其他模式“ [31 ed 5e]”来定位函数开始位置。 不幸的是，错误模式的影响太主要了。 为此，我们可以添加新样本以减少误导性功能（“ [50 fd]”）的影响，并推广正确的指标（“ [31 ed 5e]”）。 通过将十六进制值“ [50 fd]”替换为随机十六进制值来生成新样本。 通过将新样本添加到训练数据中，我们试图减少重新训练的分类器中的错误。

Evaluation Results. 为了证明修补的有效性，我们对所有5个分类器执行了上述步骤。 对于每个false positive and false negative，我们分别生成kp和kn个新样本。 注意，kp和kn不一定相同，但是它们都必须很小。 毕竟，我们希望在不损害分类器本来就很准确的情况下修补目标错误。 对于所有分类器，我们一致地替换了前5个具有误导性的功能，并以40个时期重新训练了模型。 表6显示了修补前后分类器的性能。 我们已经测试了参数的敏感性，发现只要将kp和kn设置在2到10之间（附录F），结果就保持相对一致。 由于篇幅所限，表6仅列出了每个分类器的一组结果。 我们的实验表明，对所有五个分类器进行重新训练后，误报率和误报率都可以减少。 这些结果表明，通过了解模型的行为，我们可以识别模型的弱点并相应地增强模型。

Application	Num. of Samples	k_n	k_p	Before		After	
				FN	FP	FN	FP
Binary O0	4,891,200	5	5	3	1	0	0
Binary O1	4,001,820	3	4	48	33	23	29
Binary O2	4,174,000	4	5	107	129	59	62
Binary O3	5,007,800	2	5	83	41	15	39
PDF Malware	3,000	6	15	28	13	10	5

Table 6: Classification result before and after patching. k_n (k_p) referes to the number of augmented samples generated for each false negative (false positive). Note that for function start detection, the number of samples refers to the number of total hex code in the testing set.

DISCUSSION

Benefits v.s. Risks. LEMNA旨在基于深度学习构建一个帮助安全分析师理解、审查甚至修补的安全系统。 从防御角度进行设计时，攻击者可能会使用它来寻求深度学习分类器的弱点。 但是，我们认为这不应降低LEMNA的价值，也不应成为不开发解释工具的原因。 类比是软件模糊测试技术：虽然黑客可以使用模糊测试工具来寻找可利用的漏洞，但模糊测试技术通过促进软件测试以在软件发布之前发现并修复漏洞而极大地受益于软件行业。

LEMNA输出的分析准则。 LEMNA为每个测试用例输出一个“解释”。 为了彻底检查分类器，开发人员可能需要通过LEMNA运行大量测试用例。 手动阅读每个案例的解释很费时，因此我们建议一种更有效的方法，该方法是首先将类似的解释分组。 在§6中，我们将完全相同的解释分组，然后选择“最具代表性”的案件。 在实践中，开发人员可以根据需要使用任何其他聚类技术对说明进行分组。

Broader Security Applications. LEMNA使用两个流行的安全应用程序进行评估。 还有许多其他安全应用程序，例如检测二进制代码的“函数端”，查明功能类型并检测易受攻击的代码。 鉴于他们的深度学习架构是RNN或MLP，他们也可能从LEMNA中受益。 请注意，像CNN这样的模型与MLP有一些相似之处，因此LEMNA可以潜在地帮助相关应用程序（例如图像分析）。 未来的工作将探索LEMNA在更广泛的应用领域中的适用性。

Other Deep Learning Architectures。 除了MLP和RNN，还有其他深度学习架构，例如序列到序列网络和混合网络。 尽管这些架构主要在机器翻译和图像字幕等领域获得成功，但初步证据表明它们具有在安全性中发挥更大作用的潜力。 一旦将来构建了具体的安全应用程序，我们计划在这些新架构上测试LEMNA。

Feature Obfuscation（特征混淆）。 当特征可以解释时，LEMNA很有用，但并非对所有应用程序都适用。。 特别是，研究人员最近提出了各种方法来模糊输入特征（对抗神经网络），以增加进行对抗性攻击的难度。 可能是因为特征混淆常常会降低分类器的准确性，所以这些技术尚未得到广泛

的应用。 LEMNA不适用于经过模糊特征训练的分类器。 但是，如果模型开发人员在原始功能和模糊功能之间建立了映射，则开发人员仍可以将LEMNA的输出转换为可解释的功能。

OTHER RELATED WORK

由于大多数相关作品已在§2和§3中进行了讨论，因此我们在此简要讨论其他相关作品。

Improving Machine Learning Robustness 提高机器学习的鲁棒性。 深度学习模型可以通过对抗性样本（即旨在造成误分类的恶意输入）来欺骗[61]。 为了提高模型抵抗力，研究人员提出了各种防御方法。 最相关的工作是对抗训练。 对抗训练旨在将对抗性示例添加到训练数据集中以重新训练更健壮的模型。 有多种技术可用于制作对抗性示例以进行对抗性训练。 我们的修补方法与标准对抗训练之间的主要区别在于，我们的修补基于对错误的理解。 我们尝试避免盲目地重新训练模型，而这可能会引入新的漏洞。

Mitigating the Influence of Contaminated Data. 减轻污染数据的影响。 最近的研究已经探索了减轻受污染的培训数据引起的错误分类的方法。 代表性的方法是“机器学习”，该方法通过将标准训练算法转换为求和形式来消除某些训练数据的影响。 最近的一项工作提出利用影响函数来识别导致错误分类的数据点。 我们的方法是对于现有工作：我们建议增加训练数据以修复训练不足的组件（而不是删除不良的训练数据）。 更重要的是，LEMNA帮助人类分析人员在修补错误之前先了解这些错误。

CONCLUSION

本文介绍了LEMNA，这是一种为安全应用中的各个分类结果导出高精度解释的新方法。 LEMNA将目标深度学习模型视为黑盒，并通过fused-lasso增强的混合回归模型来近似其决策边界。 通过在两个流行的基于深度学习的安全应用程序上对其进行评估，我们表明所提出的方法产生了高度准确的解释。 此外，我们演示了机器学习开发人员和安全分析师如何从LEMNA中受益，以更好地了解分类器行为，解决分类错误，甚至执行自动补丁以增强原始深度学习模型。

参考翻译与解析：

- 论文翻译：https://www.inforsec.org/wp/?p=2866
- https://www.4hou.com/posts/o6J3
- http://www.arkteam.net/?p=4264
- youtube: https://www.youtube.com/watch?v=GX5nTSHfl84&ab_channel=AssociationforComputingMachinery%28ACM%29

Review

1. 文中有提到分类器需要提前训练好供解释器使用，那么RNN分类器训练结果的优劣是否会影响解释器的判断呢?
2. 文中的例子好像都是二分类问题，在多分类问题上是否有相关的解释办法?
3. 有对其他安全程序的相关的应用吗?

4. 在文章中对fused lasso的描述主要用于更新高斯混合模型中的mean；但是没有fused lasso没有在源代码中发现相关程序？当然也可能github的源码不完整。但是fused lasso在文中是一个较为创新的点，是否可以对其功能以及原理进行简单的描述？
    - （带惩罚的高斯混合模型：https://www.cnblogs.com/huangyc/p/10275131.html）
5. 如何观察到数据满足高斯分布？
6. 文中拟合出来的模型与决策边界有什么联系？因为Lime的决策边界是一个非线性的函数，而在X附近的原模型进行近似拟合的时候为一个简单的线性模型。

疑问

- 文章中用到的那个模型到底是属于线性的还是非线性的呢？（得到的应该是一个非线性的决策边界）多元的线性模型我觉得是
- 为什么要把（X2 + X3）合并起来？而不是aX2 + bX3 + ……

$$f(\mathbf{x}) = \beta_1 x_1 + \beta_2(x_2 + x_3) + \beta_3(x_4 + x_5) + \cdots + \beta_k x_M\,, \quad (3)$$

$$y = \sum_{k=1}^{K} \pi_k(\boldsymbol{\beta}_k \mathbf{x} + \epsilon_k)\,, \quad (4)$$

源码分析

https://github.com/nitishabharathi/LEMNA/blob/master/lemna.py

读取恶意代码与良性代码将其混合并划分为训练集合与测试集合；在做一个简单的数据清洗

```
ben  = pd.read_csv('/content/drive/My Drive/benign.csv')
mal  = pd.read_csv('/content/drive/My Drive/malicious.csv')
data = ben.append(mal, ignore_index=True)
data = shuffle(data)

y = data['class']
x = data.drop(['class'], axis=1)
x = x.drop(['filename'],axis=1)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.3,
 random_state=27)

for i in data:
    if True in data[i]:
```

```
            data[i] = list(map(lambda x: categorical_helper_fun(x), data[i]))
        feature_means.append(np.mean(data[i]))
        feature_maxs.append(max(data[i]))
```

使用Lime对比工具来进行一次训练

```
l = lime.lime_tabular.LimeTabularExplainer(np.array(x_train), feature_nam
es = feature_names, class_names=['True', 'False'],
discretize_continuous=True)
```

data_inverse函数：这个函数应该是主要用于对输入进行扰乱后生成新的样本，继而再进行解释模型的训练，对判断重要特征更有帮助。因为是黑盒子模型，因此扰乱输入再根据扰乱的输入放到训练好的RNN中得到输出再进行解释模型的训练。这就是黑盒子模型的特点，只关心输入与输出。

```
预测周围生成邻域（l, data_row, num_samples）
    对于数字特征，根据训练数据中的均值和标准差，通过从正态（0,1）进行采样并对均值居中和缩放进行逆运算来扰动它们。 对于分类特征，通过根据训练分布进行采样来扰动，并在该值与所说明的实例相同时将二进制特征设为1。
    Args:
        data_row: 1d numpy数组，对应于一行
        num_samples: 学习线性模型的邻域大小
    Returns: 一个元组(data, inverse)，其中:
        data: dense num_samples * K 矩阵，其中分类特征使用以下任一编码 0（不等于data_row中的对应值）或1.第一行是原始实例。
        inverse: 与data相同，但分类特征不是二进制，而是分类（作为原始数据）
```

先用RNN训练得到相关的网络并加载，然后使用data_inverse函数对样本进行扰乱生成新的样本输入，然后将新的样本输入放到模型中进行预测得到新的样本输出，然后把全部用于训练的的样本输入与输出放到高斯混合模型中进行拟合；拟合完成后得到的模型分别生成

```
//加载RNN训练好的网络
model = pickle.load(open('/content/drive/My Drive/classifier.sav', 'rb'))
//根据训练数据中的均值和标准差，通过正态(0,1)来进行采样并对均值进行居中和缩放来进行一定的扰动
a = data_inverse(l, x_train.iloc[0], 1000)
//EM算法与混合高斯模型（GMM）。这是一种常用的聚类算法，是一种使用后验概率准确评测其精度的方法
gm = GaussianMixture(n_components=2, covariance_type = 'diag')
//选择扰动后分类特征非二进制形式的原始数据
p_data_train = a[1]
//使用训练好的RNN网络模型进行预测
p_data_test = list(map(lambda x: x>=0.5, model.predict(p_data_train)))
//使用混合高斯模型进行训练
gm.fit(p_data_train, p_data_test)

components = gm.precisions_
//使用原始数据与训练后数据同时进行混合高斯模型的预测，对比预测结果
//predict ：  使用训练好的模型预测X中数据样本的标签
```

```
ind = []
target = gm.predict(x_train.iloc[0].values.reshape(1,-1))[0]
for i in range(len(p_data_train)):
    if gm.predict(p_data_train[i].reshape(1,-1))[0]==target:
        ind.append(i)

feature_count = np.zeros(135)
// 计算每个特征有多少个数值；数值越多越重要
for i in ind:
    feature_count = list(map(int, p_data_train[i]!=0))+feature_count
```

整个流程

其实这个解释模型整个流程大致如下：

- 首先是先有分类器与训练集
- 先扰乱训练集合的部分输入得到一些新的输入
- 加载分类器，再把新的输入放到分类器中得到新的输出
- 把总的输入和输出放到混合高斯模型中进行拟合
- 最好把拟合好的模型分别预测原数据集和扰乱输入的数据集，对比结果