Exact and Consistent Interpretation of Piecewise Linear Models Hidden behind APIs: A Closed Form Solution

Zicun Cong*, Lingyang Chu§, Lanjun Wang§, Xia Hu*, Jian Pei*
*Simon Fraser University, Burnaby, Canada
§ Huawei Technologies Canada Co., Ltd., Burnaby, Canada
Emails: {zcong, huxiah, jpei}@sfu.ca, {lingyang.chu1, lanjun.wang}@huawei.com

***Abstract*—More and more AI services are provided through APIs on cloud where predictive models are hidden behind APIs. To build trust with users and reduce potential application risk, it is important to interpret how such predictive models hidden behind APIs make their decisions. The biggest challenge of interpreting such predictions is that no access to model parameters or training data is available. Existing works interpret the predictions of a model hidden behind an API by heuristically probing the response of the API with perturbed input instances. However, these methods do not provide any guarantee on the exactness and consistency of their interpretations. In this paper, we propose an elegant closed form solution named `OpenAPI` to compute exact and consistent interpretations for the family of Piecewise Linear Models (PLM), which includes many popular classification models. The major idea is to first construct a set of overdetermined linear equation systems with a small set of perturbed instances and the predictions made by the model on those instances. Then, we solve the equation systems to identify the decision features that are responsible for the prediction on an input instance. Our extensive experiments clearly demonstrate the exactness and consistency of our method.**

I. Introduction

More and more machine learning systems are deployed as cloud services to make important decisions routinely in many application areas, such as medicine, biology, financial business, and autonomous vehicles [14]. As more and more decisions in both number and importance are made, the demand on clearly interpreting these decision making processes is becoming ever stronger [16]. Accurately and reliably interpreting these decision making processes is the key to many essential tasks, such as detecting model failures [1], building trust with public users [34], and preventing models from unfairness [48].

Many methods have been proposed to interpret a machine learning model (see Section II for a brief review). Most of those methods are applicable only when they have full access to training data and model parameters. Unfortunately, they cannot interpret decisions made by machine learning models encapsulated by cloud services, because service providers always protect and hide their sensitive training data and predictive models as top commercial secrets [44]. More often than not, only application program interfaces (APIs) are provided to public users.

The local perturbation methods [7], [11], [34], [35] are developed to interpret predictive models that only APIs but no training data or model parameters are known. The major idea is to identify the decision features of a model by analyzing the predictions on a set of perturbed instances that are generated by perturbing (i.e., slightly modifying) the features of an instance to be interpreted. However, since the space of possible feature perturbations is exponentially large with respect to the dimensionality of the feature space, those methods can only heuristically search a tiny portion of the perturbation space in a reasonable amount of time. There is no guarantee that the decision features found are exactly the decision features of the model to be interpreted [8]. The reliability of the explanations still remains an unsolved big challenge [10]. Poor interpretations may mislead users in many scenarios [35].

Can we compute exact and consistent interpretations of decisions made by predictive models hidden behind cloud service APIs? In this paper, affirmatively we provide an elegant closed form solution for the family of piecewise linear models. Here, a **piecewise linear model (PLM)** is a non-linear classification model whose classification function is a piecewise linear function. In other words, a PLM consists of many locally linear regions, such that all instances in the same locally linear region are classified by the same locally linear classifier [8]. The family of PLM hosts many popular classification models, such as logistic model trees [24], [42] and the entire family of piecewise linear neural networks [8] that use MaxOut [15] or ReLU family [13], [19], [29] as activation functions. For example, the implementations of the AlexNet [23], the VGG Net [40], and the ResNet [20] all belong to the family of PLM. Due to the extensive applications [26] and tremendous practical successes [23] of piecewise linear models, exact interpretations of piecewise linear models hidden behind APIs are greatly useful in many critical application tasks.

Our major technical contribution in this paper is to develop `OpenAPI`, a method to exactly interpret the predictions made by a PLM model behind an API without accessing model parameters or training data. Specifically, `OpenAPI` identifies the decision feature, which is a vector showing the importance degree of each feature, for an instance to be interpreted by finding the closed form solutions to a set of overdetermined linear equation systems. The equation systems are simply constructed using a small set of sampled instances. We prove that the decision features identified by `OpenAPI` are exactly the decision features of the PLM with probability 1. Our interpretations are consistent within each locally linear region, because `OpenAPI` accurately identifies the decision features of a locally linear classifier, and those decision features are the same for all instances in the same locally linear region. We conduct extensive experiments to demonstrate the exactness and consistency of our interpretations superior to five state-of-the-art interpretation methods [7], [34], [38], [39], [43].

The rest of the paper is organized as follows. We review related works in Section II, and formulate our problem in Section III. We develop `OpenAPI` in Section IV, and present the experimental results in Section V. We conclude the paper in Section VI.

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00059

IEEE computer society

II. Related Works

How to interpret decisions made by predictive models is an emerging and challenging problem. There are four major groups of methods, briefly reviewed here.

First, the **instance attribution methods** find the training instances that significantly influence the prediction on an instance to be interpreted. Wojnowicz *et al.* [45] used influence sketching to identify the training instances that strongly affect the fit of a regression model by efficiently estimating Cook's distance [9]. Koh *et al.* [22] proposed influence functions to trace the prediction of a model and identify the training instances that are the most responsible for the prediction. Bien *et al.* [4] proposed a prototype selection algorithm to find a small set of representative training instances that capture the full variability of a class without confusing with the other classes. Zhou *et al.* [49] identified the instances that dominate the activation of the same hidden neuron of a convolutional neural network, and used the common labeled concept of those instances to interpret the semantic of the hidden neuron.

The instance attribution methods rely heavily on training data, which, unfortunately, is unavailable in most of the practical scenarios where only the APIs of the predictive models are provided.

Second, the **model intimating methods** train a self-explaining model to intimate the predictions of a deep neural network [3], [6], [21]. Hinton *et al.* [21] proposed to distill the knowledge of a large neural network by training a smaller network to imitate the predictions of the large network. To make the distilled knowledge easier to understand, Frosst *et al.* [12] extended the distillation method [21] by training a soft decision tree to mimic the predictions of a deep neural network. Ba *et al.* [3] trained a shallow mimic network to distill the knowledge of one or more deep neural networks. Wu *et al.* [46] used a binary decision tree to mimic and regularize the prediction function of a deep time-series model. Guo *et al.* [18] trained a Dirichlet Process regression mixture model to approximate the decision boundary of the intimated model near an instance to be interpreted.

The model intimating methods produce understandable interpretations. They, however, cannot be directly applied to interpret models hidden behind APIs, because they cannot access training data to conduct mimic training. Moreover, since a mimic model is not exactly the same as the intimated model, the interpretations may not exactly match the real behavior of the intimated model [8].

Third, the **gradient analysis methods** [39], [43], [50] find the important decision features for an instance to be interpreted by analyzing the gradient of the prediction score with respect to the instance. Simonyan *et al.* [39] generated a class-saliency map and a class-representative image for each class of instances by computing the gradient of the class score with respect to an input instance. Zhou *et al.* [50] proposed CAM to find discriminative instance regions for each class using the global average pooling in Convolutional Neural Networks (CNN). Selvaraju *et al.* [36] generalized CAM [50] to Grad-CAM by identifying important regions of an image, i.e., a sub-matrix, and propagating class-specific gradients into the last convolutional layer of a CNN. Smilkov *et al.* [41] proposed SmoothGrad to visually sharpen the gradient-based sensitivity map of an image to be interpreted. Chu *et al.* [8] transformed a piecewise linear neural network into a set of locally linear classifiers, and interpreted the prediction on an input instance by analyzing the gradients of all neurons with respect to the instance.

The interpretations produced by the gradient analysis methods are faithful to the real behavior of the model to be interpreted. The computation of gradients, however, requires full access to model parameters, which is usually not provided by the predictive models hidden behind APIs.

Last, the **local perturbation methods** interpret the behavior of a predictive model in a small neighborhood of the instance to be interpreted. The key idea is to use a simple and interpretable model to analyze the predictions on a set of perturbed instances generated by perturbing the features of the instance to be interpreted. Ribeiro *et al.* [34] proposed `LIME` to capture the decision features for an instance to be interpreted by training a linear model that fits the predictions on a sample of the perturbed instances. They also proposed `Anchors` [35] to find the explanatory rules that dominate the predictions on a sample of perturbed instances. Fong *et al.* [11] proposed to interpret the classification result of an image by finding the smallest pixel-deletion mask that causes the most significant drop of the prediction score.

The local perturbation methods, on the one hand, generate interpretations easy to understand without accessing model parameters or training data. On the other hand, the interpretations may not be even correct, since the interpretation error is proportional to $f(\varepsilon, n) + g(m)$, where the first component $f(\varepsilon, n)$ represents the *parameter related error*, ε being the perturbation distance and n the number of perturbed instances, and the second component $g(m)$ is the *approximate model related error* of the approximate model m. Parameters not well selected may lead to a large error $f(\varepsilon, n)$. The perturbation distances may be so large that the target model's behaviors on those perturbed instances are too complicated to be learned by a simple model. The *approximate model related error* is due to the weaker approximation capabilities of simple models. For example, a linear model cannot exactly describe the non-linear behavior of a target model.

Although existing methods can decrease the errors in their interpretations using smaller neighborhoods, more perturbed instances, and better approximate models, the errors cannot be eliminated due to the following reason. Different instances may have different applicable perturbation distances, that is, radii where the same interpretations still apply. The proper perturbation distance for an instance can be arbitrarily small, as the instance can be arbitrarily close to the boundary of the locally linear regions.

Figure 1 elaborates the subtlety. Suppose the 2-dimensional input space is separated into two regions by a PLM (the solid boundary). Each region has a unique linear classifier whose decision boundaries are the dashed lines. The two red solid boxes of the same size represent the neighborhoods of two instances, A and B. As the neighborhood of A completely falls into a class region of the PLM, the PLM behaves linearly there. Thus, the existing methods can obtain accurate interpretations for the prediction on **A** by applying a simple model to analyze the perturbed instances from the neighborhood. However, the neighborhood of B overlaps the decision boundary and thus the PLM does not behave linearly in the neighborhood of B. Consequently, the existing methods cannot find a simple model performing exactly the same as the PLM.

The existing methods rely on a user defined perturba-

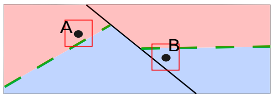


Figure 1: The hardness of getting exact interpretations for PLMs.

tion distance. However, without accessing the parameters of a target model, it is impossible to find a universally applicable perturbation distance. One may wonder whether we can shrink the neighborhood size adaptively until the approximate models perfectly fit the perturbed instances. Unfortunately, the numerical optimization techniques used to train the approximate models, such as gradient descent, do not allow the current methods to reach the exact solutions [17]. The fact that existing methods cannot guarantee exactness of interpretations prevents them from being trusted by users. When the internal parameters of a target model are unavailable, users cannot verify the correctness of the interpretations. Therefore, users cannot tell whether an unexpected explanation is caused by the misbehavior of the model or by the limitations of the explanation methods [10].

In this paper, we develop `OpenAPI` to overcome the shortage. `OpenAPI` guarantees to find the exact decision features of the model to be interpreted with probability 1, and thus leads to a significant advantage on producing exact and consistent interpretations for the PLMs hidden behind APIs.

III. Problem Definition

Denote by $\mathcal{N}$ a piecewise linear model (PLM), and by $\mathbf{x} \in \mathcal{X}$ an **input instance** of $\mathcal{N}$, where $\mathcal{X} \in \mathbb{R}^d$ is a d-dimensional input space. $\mathbf{x}$ is also called an **instance** for short. The **output** of $\mathcal{N}$ is $\mathbf{y} \in \mathcal{Y}$, where $\mathcal{Y} \in \mathbb{R}^C$ is a C-dimensional output space, and C is the total number of classes.

A PLM works as a piecewise linear classification function $F : \mathcal{X} \to \mathcal{Y}$ that maps an input $\mathbf{x} \in \mathcal{X}$ to an output $\mathbf{y} \in \mathcal{Y}$. Denote by $\mathcal{X}_k \subset \mathcal{X}$ the k-th **locally linear region** of $\mathcal{X}$ such that $F(\cdot)$ operates as a locally linear classifier in $\mathcal{X}_k$.

Denote by K the number of all locally linear regions of $F(\cdot)$. Then, $\{\mathcal{X}_1, \ldots, \mathcal{X}_K\}$ forms a partition of $\mathcal{X}$, that is, $\cup_{k=1}^{K} \mathcal{X}_k = \mathcal{X}$, and $\mathcal{X}_k \cap \mathcal{X}_h = \emptyset$ when $k \neq h$. For common PLMs such as logistic model trees [24], [42] and piecewise linear neural networks [8], [28], [31], the number of locally linear regions is finite.

Without loss of generality, we write the locally linear classifier in $\mathcal{X}_k$ as

$$\sigma(W_k^\top \mathbf{x} + \mathbf{b}_k),$$

where $W_k \in \mathbb{R}^{d \times C}$ is a d-by-C dimensional coefficient matrix of $\mathbf{x} \in \mathcal{X}_k$, $\mathbf{b}_k \in \mathbb{R}^C$ is a C-dimensional bias vector, and $\sigma(\cdot)$ is a probabilistic scoring function, which can be sigmoid and softmax for binary classification and multi-class classification, respectively. Since the softmax function is the general form of the sigmoid function, we assume $\sigma(\cdot)$ to be the softmax function by default, and write the complete form of $F(\cdot)$ as follows.

$$F(\mathbf{x}) = \begin{cases} \sigma(W_1^\top \mathbf{x} + \mathbf{b}_1) & \text{if } \mathbf{x} \in \mathcal{X}_1 \\ \sigma(W_2^\top \mathbf{x} + \mathbf{b}_2) & \text{if } \mathbf{x} \in \mathcal{X}_2 \\ \qquad \vdots & \\ \sigma(W_K^\top \mathbf{x} + \mathbf{b}_K) & \text{if } \mathbf{x} \in \mathcal{X}_K \end{cases}$$

Given an input instance $\mathbf{x}$, without loss of generality, denote by $\mathcal{X}_k$ the locally linear region that contains $\mathbf{x}$, the classification on $\mathbf{x}$ is uniquely determined by the locally linear classifier $\sigma(W_k^\top \mathbf{x} + \mathbf{b}_k)$. For the sake of simplicity, we omit the subscript k when it is clear from the context, and write the classification result of $\mathbf{x}$ as

$$\mathbf{y} = \sigma(W^\top \mathbf{x} + \mathbf{b})$$

Following a principled approach of interpreting a machine learning model [5], [7], [34], we regard an interpretation on the classification result of an input instance $\mathbf{x}$ as the decision features that classify $\mathbf{x}$ as one class and distinguish $\mathbf{x}$ from the other $C-1$ classes. The formal definition of decision features will be discussed in Section IV-A. Formally, we define the task to interpret PLMs hidden behind APIs as follows.

Definition 1. *Given the API of a PLM $\mathcal{N}$ and an input instance $\mathbf{x} \in \mathcal{X}$, for each class $c \in \{1, \ldots, C\}$, our task is to identify the decision features of $\mathcal{N}$ that classify* $\mathbf{x}$ *as class c and distinguish it from the other $C-1$ classes.*

IV. Interpretation Methods

In this section, we first introduce the decision features of a PLM in classifying an instance. Then, we illustrate a naive method to compute the decision features of a PLM under an ideal case. Last, as the ideal case may not always appear, we introduce the `OpenAPI` method that computes the exact decision features without using any training data or model parameters.

A. Decision Features of a PLM

Some existing methods [2] interpret model predictions by computing the partial derivatives of model outputs with respect to input features. The partial derivatives are used as importance weights of features. However, those methods do not work well for PLMs hidden behind APIs. First, to reliably compute the exact partial derivatives, the internal parameters of PLMs are needed. Second, for all instances in the same locally linear region, the weights of the corresponding features have to be consistent, as they are classified by the same locally linear classifier [8]. However, the feature weights computed by the gradient-based methods are different for different input instances.

Based on the coefficient matrices of the locally linear classifiers, we propose a new way to interpret the predictions made by PLMs. Our proposed interpretation not only describes the behaviors of PLMs exactly but is also consistent for the predictions made by the same locally linear classifiers.

Consider the output $\mathbf{y}$ of a PLM $\mathcal{N}$ on an instance $\mathbf{x}$. For any class $c \in \{1, \ldots, C\}$, the c-th entry of $\mathbf{y}$, denoted by $\mathbf{y}_c$, is the probability to predict $\mathbf{x}$ as class c. Denote by $W_c \in \mathbb{R}^d$ the c-th column of W and by $\mathbf{b}_c \in \mathbb{R}$ the c-th entry of $\mathbf{b}$, we can expand the locally linear classifier $\mathbf{y}=\sigma(W^{\top}\mathbf{x}+\mathbf{b})$ and write the c-th entry of $\mathbf{y}$ as $\mathbf{y}_c \propto e^{W_c^{\top}\mathbf{x}+\mathbf{b}_c}$.

Following the routine of interpreting conventional linear classifiers, such as Logistic Regression and linear SVM [5], W_c is the vector of weights for all features in predicting $\mathbf{x}$ as class c. The features with positive (negative) weights in W_c support (oppose) to predict $\mathbf{x}$ as class c.

Denote by $W_{c'}$, $c' \neq c$, the vector of weights for all features in predicting $\mathbf{x}$ as class c'. The difference between W_c and $W_{c'}$, $D_{c,c'} = W_c - W_{c'}$, identifies the features that classify $\mathbf{x}$ as class c and distinguishes $\mathbf{x}$ from class c'. To be specific, as $\mathbf{y}_c/\mathbf{y}_{c'} \propto e^{(W_c-W_{c'})^{\top}\mathbf{x}+\mathbf{b}_c-\mathbf{b}_{c'}}$, the input features of positive values in $D_{c,c'}$ increase the confidence of the model on class c over class c', and vice versa. As a result, $D_{c,c'}$ defines the **decision boundary** between class c and class c', thus is exactly the decision features of binary classification PLMs.

For general multi-class classification PLMs (i.e., $C \geq 2$), we interpret their predictions in the way of one-vs-the-rest. We can identify the decision features that classify $\mathbf{x}$ as class c and distinguish it from the other $C-1$ classes by the average of the vectors $D_{c,c'}$ for all $c' \in \{1,\dots,C\} \setminus c$. Since $D_{c,c} = \mathbf{0}$, we can write this average of vectors as

$$D_c = \frac{1}{C-1}\sum_{c'=1}^{C} D_{c,c'} \tag{1}$$

Here, the **decision features** D_c are a d-dimensional vector that contains the importance weight of each feature in classifying $\mathbf{x}$ as class c. A feature with a larger absolute weight in D_c is more important than one with a smaller absolute weight in classifying $\mathbf{x}$ as class c. In addition, the signs of the weights in D_c indicate the directions of the influences of the features on the prediction. The features of positive weights in D_c support the predictions of the model on the class c over any other classes, and vice versa. In other words, D_c is the answer to interpreting why a PLM classifies an instance $\mathbf{x}$ as class c instead of some other classes. As D_c is computed solely from the coefficient matrices of the locally linear classifiers, for two instances $\mathbf{x}$ and $\mathbf{x}'$ in the same locally linear region, they have the same D_c. This property enables our method to provide consistent interpretations for predictions made on instances from the same locally linear regions.

We can easily compute D_c when the model parameters of a PLM are given. For example, D_c can be easily extracted from the model parameters of the conventional PLMs such as logistic model trees [24], [42]. For piecewise linear neural networks, there is also an existing method [8] that computes D_c in polynomial time when the model parameters are given. However, none of the above methods can be used to compute D_c when model parameters are unavailable.

B. A Naive Method

To use only the API of a PLM to compute D_c without accessing any model parameters, in this subsection, we introduce a naive method by solving $C-1$ determined linear equation systems. In an ideal case, the solution is exactly the same as D_c.

Given a tuple $(\mathbf{x},\mathbf{y})$ where $\mathbf{x} \in \mathcal{X}$ is an input instance and $\mathbf{y}=\sigma(W^{\top}\mathbf{x}+\mathbf{b})$ is the prediction on $\mathbf{x}$, our goal is to compute D_c for $\mathbf{x}$ by computing the set $\{D_{c,c'}\}$ such that $c' \in \{1,\dots,C\} \setminus c$.

For c and c', denote by $B_{c,c'} = \mathbf{b}_c - \mathbf{b}_{c'}$ the difference between bias vectors $\mathbf{b}_c$ and $\mathbf{b}_{c'}$. By decomposing the softmax function $\sigma(\cdot)$ of the locally linear classifier $\mathbf{y}=\sigma(W^{\top}\mathbf{x}+\mathbf{b})$, we have

$$\frac{\mathbf{y}_c}{\mathbf{y}_{c'}} = \frac{e^{W_c^{\top}\mathbf{x}+\mathbf{b}_c}}{e^{W_{c'}^{\top}\mathbf{x}+\mathbf{b}_{c'}}} = e^{{D_{c,c'}}^{\top}\mathbf{x}+B_{c,c'}},$$

which can be transformed into the following linear equation

$$ {D_{c,c'}}^{\top}\mathbf{x} + B_{c,c'} = \ln(\frac{\mathbf{y}_c}{\mathbf{y}_{c'}}) \tag{2}$$

Since $\mathbf{x}$, $\mathbf{y}_c$ and $\mathbf{y}_{c'}$ are known variables, Equation 2 contains $d+1$ unknown variables, which are the entries of the d-dimensional vector $D_{c,c'} \in \mathbb{R}^d$ and the scalar $B_{c,c'} \in \mathbb{R}$.

Tuple $(D_{c,c'}, B_{c,c'})$ fully characterizes the behavior of a locally linear classifier $\mathbf{y}=\sigma(W^{\top}\mathbf{x}+\mathbf{b})$ in classifying classes c and c'. If two locally linear classifiers have exactly the same $(D_{c,c'}, B_{c,c'})$ for every pair c and c', they produce exactly the same output $\mathbf{y}$ for the same input instance $\mathbf{x}$. As a result, we call $(D_{c,c'}, B_{c,c'})$ the **core parameters** of a locally linear classifier in classifying classes c and c'. The core parameters of the locally linear classifier for an instance $\mathbf{x}$ is also said to be the core parameters of $\mathbf{x}$ for short.

For any pair c and c', a **naive method** to compute the core parameters of $\mathbf{x}$ is to construct and solve a determined linear equation system, denoted by $\Omega_{d+1}^{c,c'}$, that consists of $d+1$ linearly independent linear equations with the same core parameters as $\mathbf{x}$.

Since we already obtain one of these linear equations from $(\mathbf{x},\mathbf{y})$, we only need to build another d linear equations by independently and uniformly sampling d instances in the neighborhood of $\mathbf{x}$. A d-dimension **hypercube** with edge length $2r$ and $\mathbf{x}$ as the center is defined as $\{\mathbf{p} \mid \forall i \; |\mathbf{p}_i - \mathbf{x}_i| \leq r,\; \mathbf{p} \in \mathbb{R}^d\}$, where $\mathbf{x}_i$ is the i-th entry of $\mathbf{x}$. In this paper, the **neighborhood** of $\mathbf{x}$ refers to the hypercube centered at $\mathbf{x}$. We will illustrate how to compute r later in Algorithm 1.

Denote by $\mathbf{x}^i, i \in \{1,\dots,d\}$, the i-th sampled instance in the neighborhood of $\mathbf{x}$, and by $\mathbf{y}^i$ the prediction on $\mathbf{x}^i$. Obviously, $\mathbf{y}^i$ can be easily obtained by feeding $\mathbf{x}^i$ into the API of a PLM. Tuple $(\mathbf{x}^i,\mathbf{y}^i)$ is used to build the i-th linear equation of $\Omega_{d+1}^{c,c'}$ in the same way as Equation 2.

In the **ideal case** where the core parameters of all sampled instances are the same as the core parameters of $\mathbf{x}$, all linear equations in $\Omega_{d+1}^{c,c'}$ are linear equations of the same core parameters as $\mathbf{x}$. Therefore, we can write $\Omega_{d+1}^{c,c'}$ as

$$\Omega_{d+1}^{c,c'} = \begin{cases} {D_{c,c'}}^{\top}\mathbf{x}^0 + B_{c,c'} = \ln(\frac{\mathbf{y}_c^0}{\mathbf{y}_{c'}^0}) \\ {D_{c,c'}}^{\top}\mathbf{x}^1 + B_{c,c'} = \ln(\frac{\mathbf{y}_c^1}{\mathbf{y}_{c'}^1}) \\ \qquad\vdots \\ {D_{c,c'}}^{\top}\mathbf{x}^d + B_{c,c'} = \ln(\frac{\mathbf{y}_c^d}{\mathbf{y}_{c'}^d}) \end{cases} \tag{3}$$

where $(D_{c,c'}, B_{c,c'})$ is the core parameters of $\mathbf{x}$ in classifying classes c and c', and $\mathbf{x}$, $\mathbf{y}_c$ and $\mathbf{y}_{c'}$ are rewritten as $\mathbf{x}^0$, $\mathbf{y}_c^0$ and $\mathbf{y}_{c'}^0$, respectively, for notational consistency.

Next, we prove that the linear equations in $\Omega_{d+1}^{c,c'}$ are linearly independent.

Denote by $\mathbf{x}^i_j \in \mathbb{R}$ the j-th entry of $\mathbf{x}^i$. We write the coefficient matrix of $\Omega^{c,c'}_{d+1}$ as a $(d+1)$-by-$(d+1)$ dimensional square matrix

$$A = \begin{bmatrix} 1 & \mathbf{x}^0_1 & \mathbf{x}^0_2 & \dots & \mathbf{x}^0_d \\ 1 & \mathbf{x}^1_1 & \mathbf{x}^1_2 & \dots & \mathbf{x}^1_d \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & \mathbf{x}^d_1 & \mathbf{x}^d_2 & \dots & \mathbf{x}^d_d \end{bmatrix},$$

where the first column stores the coefficients for variable $B_{c,c'}$. We prove that the linear equations in $\Omega^{c,c'}_{d+1}$ are linearly independent by showing that A is a full rank matrix with probability 1.

Lemma 1. *When the perturbed instances are independently and uniformly sampled from a hypercube, the coefficient matrix A of $\Omega^{c,c'}_{d+1}$ is a full rank matrix with probability* 1.

Proof:

Denote by $A_i \in \mathbb{R}^{d+1}, i \in \{0,\dots,d\}$, the i-th row of A, by $\overline{A_i}$ the sub-vector containing the last d entries of A_i, that is, $\overline{A_i} = [x^i_1, x^i_2, \dots, x^i_d] = x^i$. Next, we prove the lemma by contradiction.

Assume the rank of matrix A is not full. The last row of matrix A must be a linear combination of the other rows. Denote by $\alpha_0,\dots,\alpha_{d-1}$ the weights of a linear combination, we write $A_d = \alpha_0 * A_0 + \alpha_1 * A_1 + \dots + \alpha_{d-1} * A_{d-1}$.

Since the first entry of every row vector A_i is 1, $\alpha_0 + \alpha_1 + \dots + \alpha_{d-1} = 1$. Recall that $\overline{A_i}$ is a subvector of A_i for all $i \in \{0,\dots,d\}$, we have

$$\overline{A_d} = \alpha_0 * \overline{A_0} + \alpha_1 * \overline{A_1} + \dots + \alpha_{d-1} * \overline{A_{d-1}} \tag{4}$$

By plugging $\alpha_{d-1} = 1 - (\alpha_0 + \alpha_1 + \dots + \alpha_{d-2})$ into Equation 4, we can derive

$$\overline{A_d} = \alpha_0 * (\overline{A_0} - \overline{A_{d-1}}) + \dots + \alpha_{d-2} * (\overline{A_{d-2}} - \overline{A_{d-1}}) + \overline{A_{d-1}} \tag{5}$$

Since Equation 5 only contains the $d-1$ free variables $\alpha_0,\dots,\alpha_{d-2}$, $\overline{A_d}$ is contained in the $(d-1)$-dimensional subspace $\mathcal{V}$ spanned by $(\overline{A_0} - \overline{A_{d-1}})$, ..., $(\overline{A_{d-2}} - \overline{A_{d-1}})$.

Since $\overline{A_d} = x^d$, $\overline{A_d}$ is independently and uniformly sampled from a d-dimensional continuous space, and the probability that $\overline{A_d}$ is sampled from the $(d-1)$-dimensional subspace $\mathcal{V}$ is 0. Therefore, the probability that Equation 5 holds is 0, which means A_d cannot be represented as a linear combination of the other rows. This contradicts with the assumption that A is not a full rank matrix. In sum, A is a full rank matrix with probability 1. ■

Lemma 1 holds as long as the perturbed instances are independently and uniformly sampled from a hypercube. By Lemma 1, $\Omega^{c,c'}_{d+1}$ is a determined linear equation system that is guaranteed to have a unique solution with probability 1.

By solving each of the $C-1$ linear equation systems in $\{\Omega^{c,c'}_{d+1} \mid c' \in \{1,\dots,C\} \setminus c\}$, we can easily determine the core parameters of $\mathbf{x}^0$ for each pair of c and c'. Then, we can apply Equation 1 to compute D_c.

The naive method introduced above is applicable when all sampled instances and the instance $\mathbf{x}^0$ have the same core parameters. However, since we do not know the model parameters of the PLM, we cannot guarantee that those instances have the same core parameters. In other words, the ideal case may not always hold in practice. In sequel, the naive method cannot accurately compute D_c all the time. Indeed, when the ideal case assumption does not hold, the performance of the naive method can be arbitrarily bad.

Theorem 1. *Denote by β^* the solution of $\Omega^{c,c'}_{d+1}$. When the ideal case does not hold, the probability that β^* is the core parameters of $\mathbf{x}^0$ is 0 for at least one pair of classes c and c'.*

Proof: Denote by $\beta^i = (D^i_{c,c'}, B^i_{c,c'}), i \in \{0,\dots,d\}$, the core parameters of $\mathbf{x}^i$, and by $\mathbb{P}(\beta^* = \beta^0)$ the probability of $\beta^* = \beta^0$. We only need to show $\mathbb{P}(\beta^* = \beta^0) = 0$ for at least one pair of c and c'.

When the ideal case does not hold, there is at least one sampled instance, denoted by $\mathbf{x}^i, i \in \{1,\dots,d\}$, that does not have the same core parameters as $\mathbf{x}^0$. Therefore, $\beta^i \neq \beta^0$ for at least one pair of classes c and c'.

By the definition of β^i, $\mathbf{x}^i$ satisfies

$${D^i_{c,c'}}^\top \mathbf{x}^i + B^i_{c,c'} = \ln(\frac{\mathbf{y}^i_c}{\mathbf{y}^i_{c'}}).$$

If $\beta^* = \beta^0$, then $\mathbf{x}^i$ satisfies

$${D^0_{c,c'}}^\top \mathbf{x}^i + B^0_{c,c'} = \ln(\frac{\mathbf{y}^i_c}{\mathbf{y}^i_{c'}}).$$

Therefore, a necessary condition for $\beta^* = \beta^0$ is that $\mathbf{x}^i$ satisfies

$$(D^i_{c,c'} - D^0_{c,c'})^\top \mathbf{x}^i + (B^i_{c,c'} - B^0_{c,c'}) = 0. \tag{6}$$

As a result, $\mathbb{P}(\beta^* = \beta^0)$ cannot be larger than the probability P that $\mathbf{x}^i$ satisfies Equation 6.

Recall that $\beta^i \neq \beta^0$ for at least one pair of c and c'. The value of P must fall into one of the following two cases.

Case 1: if $D^i_{c,c'} = D^0_{c,c'}$, then $B^i_{c,c'} \neq B^0_{c,c'}$. In this case, no $\mathbf{x}^i$ satisfies Equation 6, thus $P = 0$.

Case 2: if $D^i_{c,c'} \neq D^0_{c,c'}$, then P is the probability that $\mathbf{x}^i$ is located on the hyperplane defined by Equation 6. In this case, P is still 0 because $\mathbf{x}^i$ is independently uniformly sampled from a d-dimensional hypercube.

In summary, $\mathbb{P}(\beta^* = \beta^0) \leq P = 0$. The theorem follows. ■

In summary, the naive method only works in the idea case where all perturbed instances have the same core parameters as the input instance $\mathbf{x}^0$. The extremely strong assumption limits the method to be usable in practice. First, as discussed in Section II, it is impossible for users to heuristically select a perturbation distance that works for all instances. Second, if the perturbed instances have different core parameters, according to Lemma 1 and Theorem 1, the naive method may not obtain a correct interpretation. Next, we develop `OpenAPI` to overcome these weaknesses.

C. The `OpenAPI` Method

Now we are ready to introduce the `OpenAPI` method to reliably and accurately compute D_c. Different from the naive method, `OpenAPI` adaptively shrinks the perturbation distance until it computes the exact interpretations with probability 1.

For any two classes c and c', `OpenAPI` computes the core parameters β^0 of $\mathbf{x}^0$ by solving an **overdetermined** linear equation system with $d+2$ linear equations. Denote by $\Omega_{d+2}^{c,c'}$ the overdetermined linear equation system. We build the first $d+1$ linear equations of $\Omega_{d+2}^{c,c'}$ in the same way as the naive method. The $(d+2)$-th linear equation of $\Omega_{d+2}^{c,c'}$ is built by sampling an extra instance $\mathbf{x}^{d+1}$ in the neighborhood of the input instance $\mathbf{x}^0$.

Denote by $\beta^i = (D_{c,c'}^i, B_{c,c'}^i)$, $i \in \{0, \ldots, d+1\}$, the core parameters of $\mathbf{x}^i$ in classifying classes c and c'. We now show that, when $\Omega_{d+2}^{c,c'}$ has at least one solution, the solution is unique and is equal to every β^i with probability 1.

Theorem 2. *For any two classes c and c', if $\Omega_{d+2}^{c,c'}$ has at least one solution, then the solution is unique and is exactly β^i for any $i \in \{0, \ldots, d+1\}$ with probability 1.*

Proof: Denote by $\Theta_i^{c,c'}$ the linear equation system that is constructed by removing the linear equation with respect to $\mathbf{x}^i$ from $\Omega_{d+2}^{c,c'}$ and keeping the rest $d+1$ linear equations. Obviously, any solution of $\Omega_{d+2}^{c,c'}$ is a solution of $\Theta_i^{c,c'}$.

According to Lemma 1, the coefficient matrix of $\Theta_i^{c,c'}$ is a full rank square matrix with probability 1. This means that the probability that $\Theta_i^{c,c'}$ has a unique solution is 1.

Since $\Omega_{d+2}^{c,c'}$ has at least one solution and any solution of $\Omega_{d+2}^{c,c'}$ is a solution of $\Theta_i^{c,c'}$, the solution of $\Omega_{d+2}^{c,c'}$ is unique, and is equal to the solution of $\Theta_i^{c,c'}$. Next, we prove that, with probability 1, the solution of $\Omega_{d+2}^{c,c'}$ is exactly β^i for any $i \in \{0, \ldots, d+1\}$.

Denote by $\beta^* = (D_{c,c'}^*, B_{c,c'}^*)$ the unique solution of $\Omega_{d+2}^{c,c'}$, and by $\mathbb{P}(\beta^* = \beta^i)$ the probability of $\beta^* = \beta^i$. We only need to show $\mathbb{P}(\beta^* \neq \beta^i) = 0$ for any $i \in \{0, \ldots, d+1\}$.

By the definition of β^i, $\mathbf{x}^i$ satisfies

$$D_{c,c'}^{i}{}^\top \mathbf{x}^i + B_{c,c'}^i = \ln(\frac{\mathbf{y}_c^i}{\mathbf{y}_{c'}^i}).$$

Since β^* is the unique solution of $\Omega_{d+2}^{c,c'}$, $\mathbf{x}^i$ also satisfies

$$D_{c,c'}^{*}{}^\top \mathbf{x}^i + B_{c,c'}^* = \ln(\frac{\mathbf{y}_c^i}{\mathbf{y}_{c'}^i}).$$

Therefore, $\mathbf{x}^i$ must satisfy

$$(D_{c,c'}^* - D_{c,c'}^i)^\top \mathbf{x}^i + (B_{c,c'}^* - B_{c,c'}^i) = 0 \qquad (7)$$

Consequently, a necessary condition for $\beta^* \neq \beta^i$ is that $\mathbf{x}^i$ is located on the hyperplane $\mathcal{H}$ defined by Equation 7. Therefore, $\mathbb{P}(\beta^* \neq \beta^i)$ cannot be larger than the probability P that $\mathbf{x}^i$ is located on $\mathcal{H}$. The value of $\mathbb{P}(\beta^* \neq \beta^i)$ must fall into one of the following three cases.

Case 1: if $D_{c,c'}^* = D_{c,c'}^i$ and $B_{c,c'}^* = B_{c,c'}^i$, then $\beta^* = \beta^i$, which means $\mathbb{P}(\beta^* \neq \beta^i) = 0$.

Case 2: if $D_{c,c'}^* = D_{c,c'}^i$ and $B_{c,c'}^* \neq B_{c,c'}^i$, then no $\mathbf{x}^i$ satisfies Equation 7, which means $P = 0$. Thus, $\mathbb{P}(\beta^* \neq \beta^i) \leq P = 0$.

Case 3: if $D_{c,c'}^* \neq D_{c,c'}^i$, because $\mathbf{x}^0$ is drawn from an underlying continuous distribution in the d-dimensional space $\mathcal{X}$ [30], and each $\mathbf{x}^i, i \in \{1, \ldots, d+1\}$, is uniformly sampled from a d-dimensional hypercube, we have $P = 0$. Therefore, $\mathbb{P}(\beta^* \neq \beta^i) \leq P = 0$.

In summary, $\mathbb{P}(\beta^* \neq \beta^i) = 0$ and the theorem follows. ■

```
Input: $\mathcal{A}$ := the API of a PLM, $c$ := the class $c$ to
        interpret, $\mathbf{x}^0$ := the instance to interpret,
        $m$ :=the maximum number of iterations.
Output: $D_c^*$ := the value of $D_c$ computed by
        OpenAPI, $r$ := the hypercube edge length.
Initialize: $r \leftarrow 1.0$, $\mathcal{I} \leftarrow \emptyset$, $D_c^* \leftarrow null$.
while $m \neq 0$ do
    Sample $d+1$ points in the hypercube with edge
      length $r$: $S \leftarrow \{\mathbf{x}^1, \ldots, \mathbf{x}^{d+1}\}$.
    for each $c' \in \{1, \ldots, C\} \setminus c$ do
        Construct $\Omega_{d+2}^{c,c'}$ by $d+2$ points in $S \cup \mathbf{x}^0$.
        If $\Omega_{d+2}^{c,c'}$ has a solution $\beta^*$ then $\mathcal{I} \leftarrow \mathcal{I} \cup \beta^*$.
    end
    if $|\mathcal{I}| < C-1$ then
        $\mathcal{I} \leftarrow \emptyset$, $r \leftarrow r/2$.
    else
        Compute $D_c^*$ from $\mathcal{I}$ by Equation 1, and
          break.
    end
    $m \leftarrow m-1$
end
return $D_c^*$, $r$.
```

Algorithm 1: `OpenAPI`$(\mathcal{A}, c, \mathbf{x}^0, m)$

According to Theorem 2, if $\Omega_{d+2}^{c,c'}$ has a solution, then it is the core parameters β^0 of $\mathbf{x}^0$ with probability 1. In this case, we can directly compute β^0 as the closed form solution to $\Omega_{d+2}^{c,c'}$. If $\Omega_{d+2}^{c,c'}$ has no solution, we can reconstruct it by randomly sampling a new set of instances in the neighborhood of $\mathbf{x}^0$, and solve the corresponding linear equation system. This iteration of reconstructions continues until we sample a set of instances that have the same core parameters as $\mathbf{x}^0$. Then, we can find the solution to $\Omega_{d+2}^{c,c'}$, which is β^0 with probability 1.

Recall that all instances within the same locally linear region have the same core parameters. If we sample instances from a **proper hypercube** that is contained in the locally linear region of $\mathbf{x}^0$, then the instances sampled certainly have the same core parameters as $\mathbf{x}^0$, and we are sure to find the valid solution β^0.

Intuitively, a hypercube with smaller edge length r is more likely to be contained by the locally linear region of $\mathbf{x}^0$. However, it is impractical to empirically set one value of r to fit all PLMs and arbitrary instances to be interpreted, because the sizes of locally linear regions vary significantly for different PLMs, and the maximum r of a proper hypercube can be arbitrarily small for an input instance that is very close to the boundary of a locally linear region. Therefore, as described in Algorithm 1, `OpenAPI` adaptively finds a proper hypercube by reducing the edge length r by half in each iteration of reconstruction.

As long as $\mathbf{x}^0$ is contained in a locally linear region, `OpenAPI` eventually can find a proper hypercube and compute a valid output, denoted by D_c^*. If $\mathbf{x}^0$ is located on the boundary of a locally linear region, then there is no proper hypercube with $r > 0$ for $\mathbf{x}^0$, and OpenAPI may fail to return a valid output. However, since the probability that $\mathbf{x}^0$ is located on the boundary of a locally linear region is 0, the probability that OpenAPI returns the valid D_c^* is still 1. To guarantee OpenAPI terminates even in the unlikely case that $\mathbf{x}^0$ is located on the boundary of a locally linear region, OpenAPI stops after a certain number of iterations, which is a system parameter. In our experiments, we set the maximum number of iterations for OpenAPI as 100. However, since the probability that $\mathbf{x}^0$ is located on the boundary of a locally linear region is 0, the non-terminating case never happened in our experiments, and OpenAPI always terminates in less than 20 iterations. If OpenAPI cannot find a proper hypercube within the maximum number of iterations, the smallest edge length r, which is constructed at the last iteration, will be returned.

Since OpenAPI adaptively finds a proper hypercube, the initial value of r has little influence on the accuracy of OpenAPI. Thus, we simply initialize it as $r = 1.0$ in our experiments.

OpenAPI has three major advantages. First, OpenAPI computes interpretations in closed form, and provides a solid theoretical guarantee on the exactness of interpretations. Second, our interpretation is consistent for all instances in the same locally linear region. This is because all instances contained in the same locally linear region have the same decision features, which are accurately identified by OpenAPI. Last, OpenAPI is highly efficient, of time complexity $O(T \cdot C(d+2)^3)$, where d and C are constants for a PLM, and T is the number of iterations of reconstruction.

V. Experiments

In this section, we evaluate the performance of OpenAPI by investigating the following four questions: (1) Can OpenAPI effectively explain model predictions? (2) Are the interpretations consistent? (3) How well are the perturbed instances being used for interpretations? (4) Are the computed interpretations exact?

To demonstrate that OpenAPI can effectively interpret the predictions of PLMs, we compare OpenAPI with four baseline interpretation methods, Saliency Maps [39], Gradient * Input [38], Integrated Gradient [43], and LIME [34]. The first three gradient-based methods [2] require to access the model parameters. LIME can interpret the predictions of PLMs with only API access.

Saliency Maps interprets a prediction by taking the absolute value of the partial derivative of the prediction with respect to the input features. Gradient * Input uses the feature-wise product between the partial derivative and the input itself as the interpretation for a prediction. Rather than computing the partial derivative of the input instance $\mathbf{x}^0$, Integrated Gradient computes the average partial derivatives when the input varies along a linear path from a baseline point to $\mathbf{x}^0$. LIME interprets the predictions of a classifier by training an interpretable model on the outputs of the classifier in a heuristically selected neighborhood of the input instance. We adopt the same experiment settings used in [37] and [8] for Integrated Gradient and LIME, respectively.

To evaluate the capability of interpretation with only API access to PLMs, in addition to the naive method discussed in Section IV-B, we design two more baselines by slightly extending ZOO [7] and LIME [34] as follows.

Data Sets	FMNIST		MNIST	
	Train	Test	Train	Test
PLNN	0.888	0.865	0.980	0.971
LMT	0.950	0.870	0.991	0.949

Table I: The training and testing accuracies of all models

ZOO is a zeroth-order approximation method approximating the gradients of functions. It first samples d pairs of instances by perturbing $\mathbf{x}^0$ back-and-forth along each axis of $\mathbb{R}^d$ for a heuristically fixed **perturbation distance** h. Then, it estimates the gradient of a model with respect to $\mathbf{x}^0$ by computing the symmetric difference quotient [25] between each pair of sampled instances. Equation 2 clearly shows that the derivative of $\ln(\frac{\mathbf{y}_c}{\mathbf{y}_{c'}})$ with respect to $\mathbf{x}$ is exactly $D_{c,c'}$. Thus, it is natural to use ZOO to estimate $D_{c,c'}$. Then D_c is computed from the estimated $D_{c,c'}$ in the same way as Equation 1.

LIME interprets predictions in the one-vs-the-rest way [34]. It is easy to extend LIME such that it uses D_c as its interpretations. Rather than training a linear model to approximate the predicted probability $\mathbf{y}_c$ of a perturbed instance, the extended LIME tries to fit $\ln(\frac{\mathbf{y}_c}{\mathbf{y}_{c'}})$ of the perturbed instances. Because of the linear relationship between an instance $\mathbf{x}$ and the corresponding value $\ln(\frac{\mathbf{y}_c}{\mathbf{y}_{c'}})$, the coefficients of the linear model are an approximation to $D_{c,c'}$. Similarly to ZOO, D_c is computed from the estimated $D_{c,c'}$. In our experiments, two types of linear regression models are used as approximators. The one using regular linear regression is called Linear Regression LIME and the one using ridge regression is called Ridge Regression LIME.

We use the published Python codes of Integrated Gradient[1], LIME[2] and ZOO[3]. The remaining algorithms are implemented using the PyTorch library [32]. All experiments are conducted on a server with two Xeon(R) Silver 4114 CPUs (2.20GHz), four Tesla P40 GPUs, 400GB main memory, and a 1.6TB SSD running Cenos 7 OS. Our source code is published at GitHub ⟨https://github.com/researchcode2/OpenAPI⟩.

We conduct all experiments across two public datasets, FMNIST [47] and MNIST [27]. FMNIST contains fashion images in 10 categories and MNIST contains images of handwritten digits from 0 to 9. Both datasets consist of a training set of 60,000 examples and a test set of 10,000 examples. We represent each of the 28-by-28 gray scale images by cascading the 784 pixel values into a 784-dimensional feature vector. The pixel values are normalized to the range $[0, 1]$.

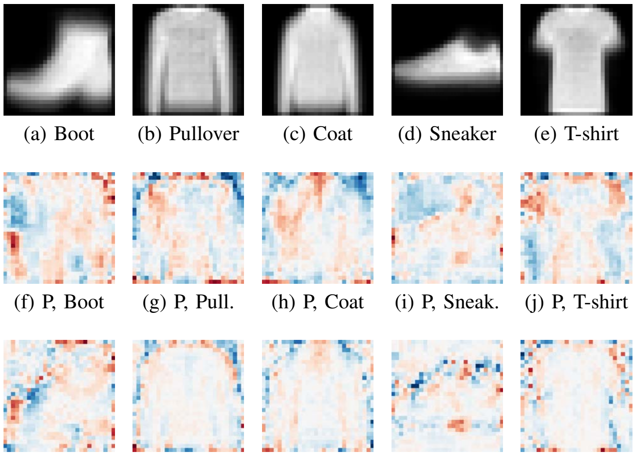
On each dataset, we train a Logistic Model Tree (LMT) [24] and a Piecewise Linear Neural Network (PLNN) [8] as the target PLMs to interpret. The classification performance of all models are shown in Table I.

Following the design in [24], we use the standard C4.5 algorithm [33] to select the pivot feature for each node and a sparse multinomial logistic regression classifier is trained

[1] https://github.com/ankurtaly/Integrated-Gradients

[2] https://github.com/marcotcr/lime

[3] https://github.com/huanzhang12/ZOO-Attack



(a) Boot (b) Pullover (c) Coat (d) Sneaker (e) T-shirt

(f) P, Boot (g) P, Pull. (h) P, Coat (i) P, Sneak. (j) P, T-shirt

(k) L, Boot (l) L, Pull. (m) L, Coat (n) L, Sneak. (o) L, T-shirt

Figure 2: The averaged images of the selected FMNIST classes and their averaged decision features of PLNN (P) and LMT (L) computed by `OpenAPI`. “Pull.” and “Sneak.” are short for pullover and sneaker, respectively

on each leaf node of the tree. To prevent overfitting, we adopt two stopping criteria. A node is not further split if it contains less than 100 training instances or the accuracy of the regression classifier is greater than 99%. Since every leaf node of a LMT is a locally linear classifier, the leaf node itself corresponds to a locally linear region, and we can directly extract the ground truth decision features for an input instance $\mathbf{x}^0$ from the multinomial logistic regression classifier of the leaf node containing $\mathbf{x}^0$.

To train a PLNN, we use the standard back-propagation to train a fully-connected network that adopts the widely used activation function ReLU [13]. The numbers of neurons from the input layer to the output layer are 784, 256, 128, 100 and 10, respectively. This network is used as a baseline model on the website ⟨https://github.com/zalandoresearch/fashion-mnist⟩ of FMNIST. We use OpenBox [8] to compute the locally linear regions and the ground truth decision features D_c for an input instance of a PLNN.

Since `LIME` is too slow to process all instances in 24 hours, for each of FMNIST and MNIST, we uniformly sample 1000 instances from the testing set, and conduct all experiments for all methods on the sampled instances.

A. Can `OpenAPI` effectively Interpret Model Predictions?

Good interpretations should be easily understood by human being. In this subsection, we first conduct a case study to illustrate the effectiveness of the interpretations. Then, we quantitatively evaluate the effectiveness of the interpretations given by `OpenAPI` and the four baseline methods. The three gradient-based methods are allowed to use the parameter information of the PLMs to compute their interpretations. `LIME` and `OpenAPI` are only allowed to use the APIs of the PLMs.

Following the tradition of interpretation visualization [2], we show the decision features as heatmaps, where red and blue colors indicate respectively features that contribute positively to the activation of the target output and features having a suppressing effect. The first row of Figure 2 shows the averaged images of five selected classes from FMNIST. For each class, its averaged decision features of the trained PLNN and LMT are shown in the second and third rows, respectively.

Comparing the heatmaps with their corresponding averaged original images, it is clear that the decision features legibly highlight the image parts with strong semantical meanings, like the heal of boots, the shoulder of pullovers, the collar of coats, the surface of sneakers, and the short sleeves of T-shirts. A closer look at the averaged images suggests that the highlighted parts describe the differences between one type of objects against the others.

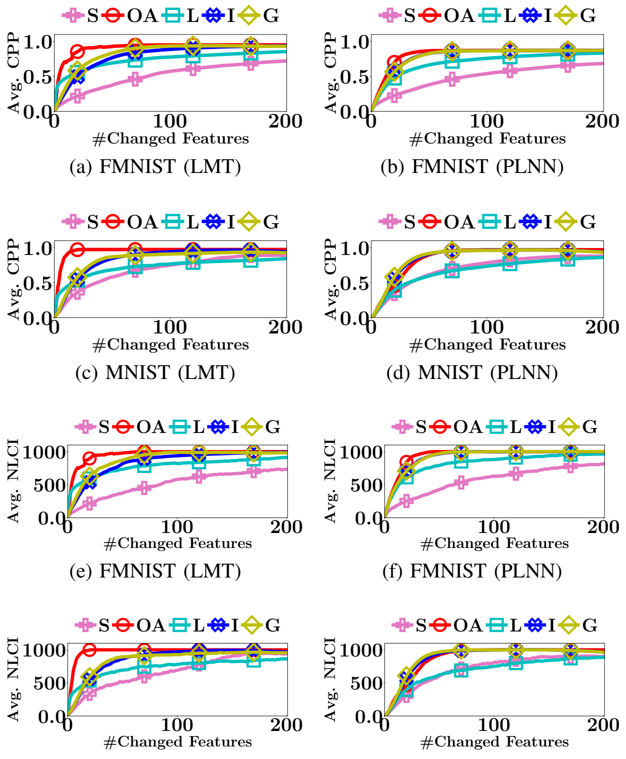
Since the LMT is trained with sparse constraints, the decision features of the LMT are sparser than the ones of the PLNN. As a result, the PLNN captures more details of the objects. Since both the LMT and the PLNN are trained on the same training data, the decision features learnt by the LMT highlight similar image patterns as the decision features of the PLNN. This demonstrates the robustness of our proposed decision features in accurately interpreting general PLMs.

To quantitatively evaluate the effectiveness of interpretations, we adopt the evaluation method used by Ancona *et al.* [2]. The method assumes that a good interpretation model should identify features that are more relevant to the predictions. Therefore, modifications on those relevant features should result in sensible variations on the predictions. Following this idea, we modify the input features according to their weights in the computed interpretations as follows.

For each interpretation method, given an input instance $\mathbf{x}^0$ with predicted label c, we sort the input features in the descending order of their absolute weights. Based on the ranking, we proceed iteratively altering the input features one at a time and up to 200 features. As the features having positive (negative) weights support (opposite) to predict $\mathbf{x}^0$ as c, to decrease the confidence of a PLM on class c, we replace the input features of positive and negative weights by 0 and 1, respectively. The changes on the predictions are evaluated by two metrics, the **change of prediction probability (CPP)** and the **number of label-changed instance (NLCI)** [8]. **CPP** is the absolute change of the probability of classifying $\mathbf{x}^0$ as c and **NLCI** is the number of instances whose predicted labels change after their features being altered.

As shown in Figure 3, `Saliency Maps` performs worst among all methods. The result is consistent with the conclusion in [2] that the instances may have features that opposite the predictions of some classes. Those features play an important role in interpreting the model predictions and can only be detected by signed interpretation methods. As shown in Figure 3 and mentioned by Ancona *et al.* [2], `Gradient * Input` captures important features better than `Integrated Gradient`. The latter involves the gradients of the unrelated instances into interpretations, therefore cannot precisely interpret the predictions. As expected, `LIME` performs poorer than most of the gradient-based methods due to the fact that `LIME` has no access to the model parameters. The lack of internal information prevents it from getting accurate interpretations. However, only with API access to the PLMs, `OpenAPI` outperforms the other methods most of the time, because our method computes the decision features that are exactly used by the

PLMs in prediction. The good performance demonstrates the effectiveness of our method.



(a) FMNIST (LMT) (b) FMNIST (PLNN) (c) MNIST (LMT) (d) MNIST (PLNN) (e) FMNIST (LMT) (f) FMNIST (PLNN) (g) MNIST (LMT) (h) MNIST (PLNN)

Figure 3: The effectiveness of different interpretation methods

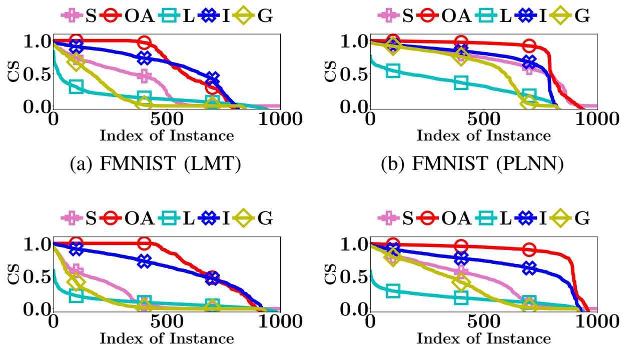
B. Are the Interpretations Consistent?

Consistent interpretation methods provide similar interpretations for similar input instances, and produce fewer contradictions between interpretations. Consistency is important. For example, it is confusing if, for the instances in a locally linear region, the weights of their corresponding features are not the same, because those instances are classified by the same locally linear classifier of the PLM.

Using the same experiment settings as Chu *et al.* [8], we comprehensively analyze the consistency of the interpretations produced by `Saliency Maps`, `Integrated Gradient`, `Gradient * Input`, and `OpenAPI` by comparing the decision features of similar input instances.

Denote by $\mathbf{x}^0$ an input instance classified as class c, and by $\mathbf{x}^1$ the testing instance that is the nearest neighbour of $\mathbf{x}^0$ in Euclidean distance. For an interpretation method, we measure the interpretation consistency by the **Cosine Similarity** (**CS**) between the computed interpretations of $\mathbf{x}^0$ and $\mathbf{x}^1$. Apparently, a larger CS indicates a better interpretation consistency.

The CS of all compared methods are evaluated on the testing data sets of FMNIST and MNIST. As shown in Figure 4, the interpretations given by `Integrated Gradient` are more consistent than the other two gradient based methods. `Integrated Gradient` smooths the differences between the interpretations for similar instances using the average partial derivatives of a set of instances to compute its interpretations. At the same time, the smooth operation also decreases the accuracy of its interpretations. The CS of `OpenAPI` is better than all other methods on all PLMs and datasets. All instances contained in the same locally linear region have exactly the same decision features, and thus the CS of `OpenAPI` should always equal to 1 on those instances. As an input instance $\mathbf{x}^0$ and its nearest neighbor in the test set may not always belong to the same locally linear region, the CS of `OpenAPI` is not equal to 1 for all instances in our experiments. The poor performances of the baseline methods can be anticipated, since their interpretations rely on the gradients of the input instances, and they tend to provide distinct interpretations for individual instances.



(a) FMNIST (LMT) (b) FMNIST (PLNN) (c) MNIST (LMT) (d) MNIST (PLNN)

Figure 4: The cosine similarity (CS for short) between the interpretations of each instance and its nearest neighbor. The results are separately sorted in the descending order of cosine similarity. “S”, “OA”, “I”, “G”, and “L” have the same meaning as in Figure 3. Each curve represents a method, and is plotted using 1000 data points. We use different markers to make the curves more legible.

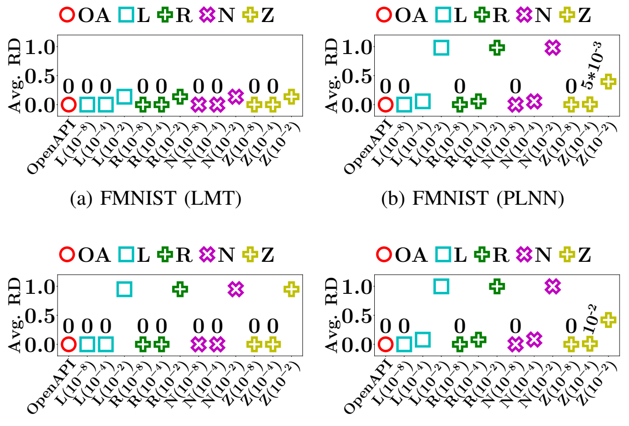
In summary, the interpretation consistency of `OpenAPI` is significantly better than the other baseline methods.

C. How Well Are the Perturbed Instances?

The accuracy of all compared methods in computing D_c of an input instance $\mathbf{x}^0$ largely depends on the **quality** of the set of sampled instances. Here, the quality of a set of instances is good if they are contained in the same locally linear region as $\mathbf{x}^0$, and thus those instances have the same core parameters as $\mathbf{x}^0$ and significantly improve the accuracy in computing D_c.

To comprehensively evaluate the performance of the compared methods in sampling a set of good instances, we measure the quality of the sampled instances by the following two metrics.

The **Region Difference (RD)** measures the consistency of the locally linear regions of the sampled instances. For any input instance $\mathbf{x}^0$, if all sampled instances are contained in the same locally linear region as $\mathbf{x}^0$, then $\mathrm{RD} = 0$; otherwise, $\mathrm{RD} = 1$.



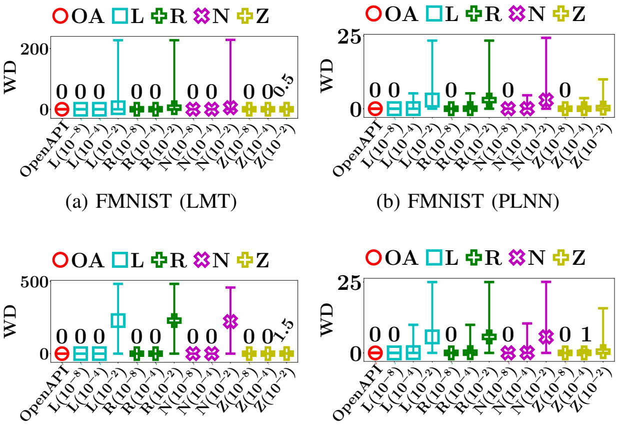
(a) FMNIST (LMT)

(b) FMNIST (PLNN)

(c) MNIST (LMT)

(d) MNIST (PLNN)

Figure 5: The average RD of all methods. $N(h)$, $Z(h)$, $L(h)$, and $R(h)$ are the performance measures of the naive method, `ZOO`, `Linear Regression LIME`, and `Ridge Regression LIME` with respect to perturbation distance h, respectively. We give the average RD values on top of some ticks for the ease of reading.



(a) FMNIST (LMT)

(b) FMNIST (PLNN)

(c) MNIST (LMT)

(d) MNIST (PLNN)

Figure 6: WD of all methods. The upper and lower ends of an error bar show the maximum and minimum WD, respectively, for all testing instances. A marker represents the mean of WD in the corresponding bar. We give the maximum of WD values on top of some ticks for the ease of reading. $N(h)$, $Z(h)$, $L(h)$, and $R(h)$ have the same meanings as in Figure 5.

The **Weight Difference (WD)** is defined as

$$\mathrm{WD} = \frac{\sum_{c'=1}^{C}\sum_{i=1}^{|S|} ||D^0_{c,c'} - D^i_{c,c'}||_1}{(C-1)|S|},$$

which measures the average L1 distance between $D^0_{c,c'}$ of the input instance $\mathbf{x}^0$ and $D^i_{c,c'}$ of the i-th instance $\mathbf{x}^i$ in the set of sampled instances $S = \{\mathbf{x}^1, \ldots, \mathbf{x}^m\}$.

Apparently, a RD that is equal to 0 indicates a perfect consistency among the locally linear regions of all sampled instances. A small WD means a high similarity between the core parameters of $\mathbf{x}^0$ and the sampled instances. If RD and WD are both small, the quality of the sampled instances is good, and D_c of $\mathbf{x}^0$ can be accurately computed.

We evaluate RD and WD of `ZOO`, `Linear Regression LIME`, `Ridge Regression LIME`, the naive method, and `OpenAPI` on the testing data sets of FMNIST and MNIST. For each data set, we use every testing instance as the input instance $\mathbf{x}^0$ once, and evaluate RD and WD of the corresponding set of sampled instances. Figures 5 and 6, respectively, show the average RD and WD of all testing instances.

The performance of the baseline methods in RD and WD relies heavily on the heuristic perturbation distance h. Since there is no effective method to set h, we evaluate the performance of the baseline methods with respect to a wide range of h. Specifically, we test $h = 10^{-2}$, 10^{-4}, and 10^{-8}.

As shown in Figure 5, the average RD of the baseline methods increases when h increases. The results verify our claim in Section IV-C that a smaller hypercube is more likely to be contained in the locally linear region of an input instance.

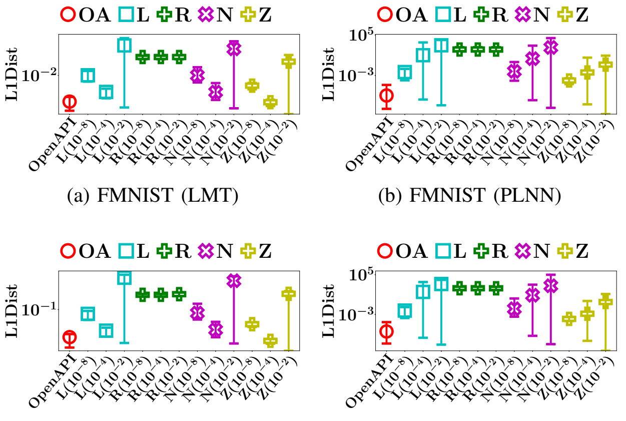
Since the RD of the baseline methods drops to 0 when h is small, it is appealing to ask whether we can fix h to a small value such that the baseline methods can always find a good sample of instances. Unfortunately, this is impossible. The volume of locally linear regions varies significantly for different PLMs. For example, as shown in Figure 5, when the perturbation distance $h = 10^{-4}$, the RD of `ZOO` is 0 for LMT, but it is 0.005 and 0.01 for PLNN on FMNIST and MNIST, respectively. Thus, $h = 10^{-4}$ is good for LMT, but not good enough for PLNN. One may argue that conservatively h can take an extremely small value in the hope that it works for both LMT and PLNN. However, since the number of locally linear regions of a PLNN is exponential with respect to the number of hidden units [8], [28], [31], the volume of some locally linear regions of a large PLNN can be arbitrarily close to zero. For any fixed value of h, one can always construct a counter example that h is still too big for PLNN. Even for the same PLM, the good perturbation distance may still vary significantly for different input instances, and can be arbitrarily small.

Recall that we can only access the API of a PLM, we have no knowledge about the size of the locally linear regions of the PLM. This makes it even harder to initialize an optimal value of perturbation distance h that works universally on all PLMs and input instances. A much better method is to sample a set of good instances in an adaptive manner, just as what `OpenAPI` does.

As shown in Figures 5 and 6, the average RD and WD of `OpenAPI` are 0 on all data sets. This demonstrates the superior capability of `OpenAPI` in adaptively sampling a set of good instances.

D. Are the Interpretations Exact?

In this subsection, we systematically study the exactness of interpretations by comparing the ground truth of the decision features of a PLM with the decision features identified by `ZOO`, `Linear Regression LIME`, `Ridge Regression LIME`, the naive method, and `OpenAPI`.



(a) FMNIST (LMT)

(b) FMNIST (PLNN)

(c) MNIST (LMT)

(d) MNIST (PLNN)

Figure 7: L1Dist of all methods. The upper and lower ends of an error bar show the maximum and minimum L1Dist, respectively, for all testing instances. A marker represents the mean of L1Dist. The L1Dist of the methods are plotted in logarithmic scale. $N(h)$, $Z(h)$, $L(h)$, and $R(h)$ have the same meaning as in Figure 5.

Denote by D_c the ground truth of decision features of a PLM in classifying an input instance as class c, and by D_c^* the decision features computed by an interpretation method. We measure the exactness of an interpretation by **L1Dist**, the L1 distance between D_c and D_c^*. Obviously, a smaller L1Dist indicates a higher exactness of an interpretation.

We evaluate L1Dist of the four baseline methods and `OpenAPI` on the testing data sets. For each data set, we use every testing instance as the input instance, and evaluate L1Dist of the interpretations. The average, minimum and maximum L1Dist of all testing instances are reported in Figure 7.

The large L1Dist of `Ridge Regression LIME` on all datasets indicates that D_c^* computed by the method is significantly different from D_c of the PLMs. By carefully investigating the learned classifiers of `Ridge Regression LIME`, we find that when the perturbed distances are very small, the linear function used to approximate the predictions always converges to a constant function that always outputs the expected value of the predictions. The poor exactness of `Ridge Regression LIME` is mainly caused by the mis-selected approximate model. As a comparison, `Linear Regression LIME`, which has no constraints on its coefficient matrix, performs much better than its counterpart with ridge regression.

The L1Dist of the other baseline methods increases significantly when the perturbation distance h becomes larger than a critical value. Since a smaller h leads to a better quality of the sampled instances, it usually increases the accuracy of most baseline methods in computing D_c^*. However, as discussed in Section V-C, the critical value of h varies significantly for different models and instances, thus it is impossible to find a golden value of h that always achieves the best L1Dist in computing the decision features of all models and instances.

We can also see that when h becomes extremely small, L1Dist increases. The reason is that all methods suffer from the classical problem of softmax saturation. When an input instance $\mathbf{x}^0$ is classified with a probability extremely close to 1 and the perturbed distance h becomes extremely small, the PLMs have almost the same predictions on the perturbed instances and the original instance. As a result, the computation of the decision features becomes unstable, which goes beyond the limited precision of Python in stably manipulating floating point numbers. Also, extremely small perturbations lead to linear equation systems with large condition numbers, which are hard to solve numerically. Due to the above reasons, extremely small perturbations hurt the exactness of all methods.

Data Set	FMNIST	MNIST
LMT	6.0	8.6
PLNN	10.3	10.8

Table II: The average number of iterations of `OpenAPI` to compute the interpretations for the models

The computation of the decision features becomes unstable due to two reasons.

In contrast, since `OpenAPI` is able to find the exact decision features of a PLM with probability 1, it achieves the best L1Dist performance on all data sets. In addition, as shown in Table II, `OpenAPI` can find the exact interpretations with only a small number of iterations.

VI. Conclusions

In this paper, we tackle the challenge of interpreting a PLM hidden behind an API. In this problem, neither model parameters nor training data are available. By finding the closed form solutions to a set of overdetermined equation systems constructed using a small set of sampled instances, we develop `OpenAPI`, a simple yet effective and efficient method accurately identifying the decision features of a PLM with probability 1. We report extensive experiments demonstrating the superior performance of `OpenAPI` in producing exact and consistent interpretations. As future work, we will extend our work to reverse engineer PLMs hidden behind APIs.

References

[1] A. Agrawal, D. Batra, and D. Parikh. Analyzing the behavior of visual question answering models. *arXiv:1606.07356*, 2016.

[2] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *ICLR*, 2018.

[3] J. Ba and R. Caruana. Do deep nets really need to be deep? In *NIPS*, pages 2654–2662, 2014.

[4] J. Bien and R. Tibshirani. Prototype selection for interpretable classification. *AOAS*, pages 2403–2424, 2011.

[5] C. Bishop. Pattern recognition and machine learning (information science and statistics). *Springer, New York*, 2007.

[6] Z. Che, S. Purushotham, R. Khemani, and Y. Liu. Distilling knowledge from deep networks with applications to healthcare domain. *arXiv:1512.03542*, 2015.

[7] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *AIS Workshop*, pages 15–26, 2017.

[8] L. Chu, X. Hu, J. Hu, L. Wang, and J. Pei. Exact and consistent interpretation for piecewise linear neural networks: A closed form solution. In *KDD*, 2018.

[9] R. D. Cook. Detection of influential observation in linear regression. *Technometrics*, 19(1):15–18, 1977.

[10] M. Du, N. Liu, and X. Hu. Techniques for interpretable machine learning. *arXiv preprint arXiv:1808.00033*, 2018.

[11] R. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. *ICCV*, 2017.

[12] N. Frosst and G. Hinton. Distilling a neural network into a soft decision tree. *arXiv:1711.09784*, 2017.

[13] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *AIS*, pages 315–323, 2011.

[14] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[15] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *ICML*, pages III–1319–III–1327, 2013.

[16] B. Goodman and S. Flaxman. European union regulations on algorithmic decision-making and a" right to explanation". *arXiv:1606.08813*, 2016.

[17] R. Grosse. Lecture note 02 in intro to neural networks and machine learning, 2018.

[18] W. Guo, S. Huang, Y. Tao, X. Xing, and L. Lin. Explaining deep learning models–a bayesian non-parametric approach. In *NIPS*, pages 4519–4529, 2018.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[21] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

[22] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. *arXiv:1703.04730*, 2017.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[24] N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine Learning*, 59(1-2):161–205, 2005.

[25] P. D. Lax and M. S. Terrell. *Calculus with Applications*. Springer, 2014.

[26] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[27] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[28] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *NIPS*, pages 2924–2932, 2014.

[29] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[30] N. M. Nasrabadi. Pattern recognition and machine learning. *JEI*, 16(4):049901, 2007.

[31] R. Pascanu, G. Montufar, and Y. Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv:1312.6098*, 2013.

[32] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS Workshop*, 2017.

[33] J. R. Quinlan. *C4. 5: Programs for Machine Learning*. Elsevier, 2014.

[34] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *KDD*, pages 1135–1144, 2016.

[35] M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI*, 2018.

[36] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *arXiv:1610.02391*, 2016.

[37] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. *PMLR*, 2017.

[38] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[39] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv:1312.6034*, 2013.

[40] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[41] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. Smoothgrad: Removing noise by adding noise. *arXiv:1706.03825*, 2017.

[42] M. Sumner, E. Frank, and M. Hall. Speeding up logistic model tree induction. *DMKD*, pages 675–683, 2005.

[43] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. *PMLR*, 2017.

[44] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In *USS*, pages 601–618, 2016.

[45] M. Wojnowicz, B. Cruz, X. Zhao, B. Wallace, M. Wolff, J. Luan, and C. Crable. "influence sketching": Finding influential samples in large-scale regressions. In *ICBD*, pages 3601–3612, 2016.

[46] M. Wu, M. C. Hughes, S. Parbhoo, M. Zazzi, V. Roth, and F. Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. In *AAAI*, 2018.

[47] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[48] R. Zemel, Y. Wu, K. Swersky, T. Pitassi, and C. Dwork. Learning fair representations. In *ICML*, pages 325–333, 2013.

[49] B. Zhou, D. Bau, A. Oliva, and A. Torralba. Interpreting deep visual representations via network dissection. *TPAMI*, 2018.

[50] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *CVPR*, pages 2921–2929, 2016.