Honest-but-Curious Nets: Sensitive Attributes of Private Inputs can be Secretly Coded into the Entropy of Classifiers' Outputs

Mohammad Malekzadeh
Imperial College London
m.malekzadeh@imperial.ac.uk

Anastasia Borovykh
Imperial College London
a.borovykh@imperial.ac.uk

Deniz Gündüz
Imperial College London
d.gunduz@imperial.ac.uk

ABSTRACT

It is known that deep neural networks, trained for the classification of a non-sensitive target attribute, can reveal sensitive attributes of their input data; through features of different granularity extracted by the classifier. We, taking a step forward, show that deep classifiers can be trained to secretly encode a sensitive attribute of users' input data, at inference time, into the classifier's outputs for the target attribute. An attack that works even if users have a white-box view of the classifier, and can keep all internal representations hidden except for the classifier's estimation of the target attribute. We introduce an information-theoretical formulation of such adversaries and present efficient empirical implementations for training honest-but-curious (HBC) classifiers based on this formulation: deep models that can be accurate in predicting the target attribute, but also can utilize their outputs to secretly encode a sensitive attribute. Our evaluations on several tasks in real-world datasets show that a semi-trusted server can build a classifier that is not only perfectly honest but also accurately curious. Our work highlights a vulnerability that can be exploited by malicious machine learning service providers to attack their user's privacy in several seemingly safe scenarios; such as encrypted inferences, computations at the edge, or private knowledge distillation. We conclude by showing the difficulties in distinguishing between standard and HBC classifiers and discussing potential proactive defenses against this vulnerability of deep classifiers.

1 INTRODUCTION

A machine learning (ML) classifier, trained on a set of labeled data, aims to facilitate the estimation of a *target* label (*i.e., attribute*) for new data at inference time; from smile detection for photography [75] to automated detection of a disease on medical data [11, 17]. However, in addition to the target attribute that the classifier is trained for, data might also contain some other *sensitive* attributes. For example, there are several attributes that can be inferred from a face image; such as gender, age, race, emotion, hairstyle, and more [32, 79]. Since ML classifiers, particularly deep neural networks (DNNs), are becoming popular, either as cloud-based services or as part of apps on our personal devices, it is important to become aware of the type of sensitive attributes that we might reveal through using these classifiers; especially when a classifier is supposed to only estimate a specified attribute. In multi-party computation, a legitimate party that does not deviate from its specified protocol but will attempt to infer as much sensitive information as possible from the received data is called a *honest-but-curious* (HBC) party [8, 15, 28, 49]. Following convention, if a classifier's *output* allows not only to estimate the target attribute, but also reveals information about other attributes (particularly those uncorrelated to the target one) we call it an HBC classifier.

In this paper, we show how a semi-trusted server can train an HBC classifier such that the output of the classifier is not only useful for inferring the target attribute, but can also secretly carry information about a sensitive attribute of the user's data that is unrelated to the target attribute. An overview of this problem is shown in Figure 1. We consider a *server* that provides its *users* access to a classifier trained for a known target task y. We put no restriction on the input data $\mathbf{x}$ or the users' access to the classifier: users can have control over the classifier and get a white-box view to it (*e.g.,* when the classifier is deployed on the users' devices), or users can perform secure computation on their data if they have a black-box view (*e.g.,* when the classifier is hosted in the cloud). Our only assumption is that both the user and the server can observe the output vector of classifier $\hat{\mathbf{y}}$. For a Y-class classifier, the output is a real-valued vector of size Y containing either (1) *raw* scores $\hat{\mathbf{y}}^{\mathrm{R}} \in \mathbb{R}^{Y}$, or

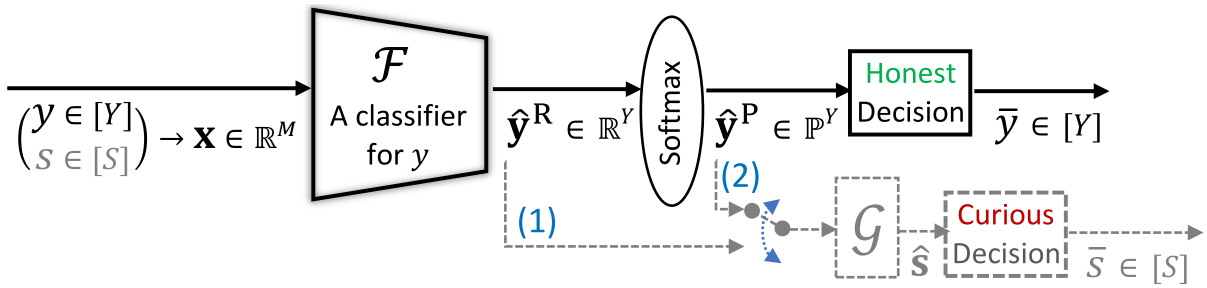

Figure 1: A *user*'s data x (*e.g.,* a face image) contains a *target* attribute y (*e.g.,* age) and a *sensitive* attribute s (*e.g.,* race). A *classifier* $\mathcal{F}$ is provided by a *server* to estimate y, and user only releases the classifier's output $\hat{\mathbf{y}}$. We show that $\mathcal{F}$ can be trained such that $\hat{\mathbf{y}}$ can not only be accurate for y (*honesty*), but can also be used by a secret attack $\mathcal{G}$ to infer s (*curiosity*). We propose efficient attacks in two scenarios: (1) the *raw* output $\hat{\mathbf{y}}^{\mathrm{R}}$ is released, or (2) the *soft* output $\hat{\mathbf{y}}^{\mathrm{P}}$ is released.

(2) *soft* outputs $\hat{\mathbf{y}}^{\mathrm{P}} = softmax(\hat{\mathbf{y}}^{\mathrm{R}}) \in \mathbb{P}^Y$ (*i.e.,* a probability mass function). For example, we show that a single real-valued output $\hat{y} \in [0, 1]$ produced by a smile-detection classifier, as the probability of "smiling", can also be used to secretly find out whether that person is "white" or not.

To protect the *user*'s privacy against an HBC classifier, it is usually proposed to hide the data, as well as all the intermediate computations on the data, and only release the classifier's output; either using secure two-party computation via cryptography [2, 3, 14] or by restricting the computations to local devices [40, 64]. Although these encrypted or local solutions hide the input data and the internal features extracted by the classifier, the output is usually released to the service provider, because the estimation of a target attribute might not seem sensitive to the user's privacy and it might be needed for further services offered to the user. Moreover, in classification tasks, an estimated probability over possible classes is more useful than just receiving the most probable class; as it allows users to aggregate estimations provided by multiple services to enhance their ultimate decision.

We first show that in a *black-box* view, where an arbitrary architecture can be used for the classifier, the server can obtain the best achievable trade-off between *honesty* (*i.e.,* the classification accuracy for the target attribute) and *curiosity* (*i.e.,* the classification accuracy for the sensitive attribute). Specifically, we build a controlled synthetic dataset and show how to create such an HBC classifier via a weighted mixture of two separately trained classifiers, one for the target attribute and another for the sensitive attribute (Section 3). Then, we focus on the more challenging, *white-box* view where the server might have some constraints on the chosen model, *e.g.,* the restriction to not being suspicious, or that the classifier must be one of the known off-the-shelf models (Section 4). To this end, we formulate the problem of training an HBC classifier in a general information-theoretical framework, via the *information bottleneck* principle [65], and show the existence of a general attack for encoding a desired sensitive data into the output of a DNN classifier. We propose two practical methods that can be used by a server for building an HBC classifier, one through *regularizing* classifier's loss function, and another via training of a *parameterized* model.

Extensive experiments, for several tasks with different attributes defined on two real-world datasets [32, 79] using a typical DNN, show that HBC classifiers can mostly achieve honesty very close to standard classifiers, while also being very successful in their curiosity (Section 5). We, theoretically and empirically, show that the entropy of an HBC classifier's output usually tends to be higher than the entropy of a standard classifier's output. Moreover, we explain how a server can improve the honesty of classifier by trading some curiosity via adding an entropy minimization component to parameterized attacks, which in particular, can make HBC classifiers less suspicious against proactive defenses [26].

Previous works propose several types of attacks *to* ML models [9, 38], mostly to DNNs [31, 39], including property inference [37], membership inference [54, 58], model inversion [13], model extraction [23, 67], adversarial examples [62], or model poisoning [6, 24]. But these attacks were mostly concerning the privacy of the training dataset and, in all these attacks, the ML model is supposed the trusted one, and users are assumed untrusted. Our work takes a different point of view and discusses a new threat model, where the (owner of) ML model is the semi-trusted party that may attack the privacy of its users at inference time. The closest related work is the *"overlearning"* concept in [61], where it is shown that internal representations extracted by DNN layers can reveal sensitive attributes of the input data that might not even be correlated to the target attribute. The assumption of [61] is that an adversary observes a subset of internal representation, while we assume all internal representations to be hidden and an adversary has access only to the output. Notably, we show that when users only release the classifier's output overlearning is not a major concern as standard classifiers do not reveal significant information about a sensitive attribute, whereas an HBC classifier can secretly, and almost perfectly, reveal a sensitive attribute just via classifier's output.

Contributions. In summary, this paper proposes the following contributions to advance privacy protection in using ML services. (1) We show that ML services can attack their user's privacy even in a highly-restricted setting where they can only get access to the results of an agreed target computation on their user's data. (2) We model such an attack in a general information-theoretical formulation and show the efficiency of our proposed model via several empirical results. Mainly, we show that since HBC classifiers encode the sensitive attributes of input data into the classifier's output, they tend to produce higher-entropy outputs than standard classifiers. However, we also show that the output's entropy can be efficiently reduced by trading a small amount of curiosity of the classifier, thus making it even harder to distinguish standard and HBC classifiers. (3) We show a real threat of this vulnerability in a recent approach where knowledge distillation [20] is used to train a student classifier on private unlabeled data via a teacher classifier that is already trained on a set of labeled data, and show that an HBC teacher can transfer its curiosity capability to the student classifiers. (4) We support our findings via several experimental results on two real-world datasets with different characteristics, as well as additional analytical results for the setting of training a convex classifiers. Code and instructions for reproducing reported results are available at https://github.com/mmalekzadeh/honest-but-curious-nets.

2 PROBLEM FORMULATION

Notation. We use lower-case *italic*, *e.g.,* x, for scalar variables; upper-case *italic*, *e.g.,* X, for scalar constants; lower-case bold, *e.g.,* $\mathbf{x}$, for vectors; upper-case blackboard, *e.g.,* $\mathbb{X}$, for sets; and calligraphic font, *e.g.,* $\mathcal{X}$, for functions; subscripts, *e.g.,* w_1, for indexing the values in a vector; superscripts, *e.g.,* w^1, for distinguishing different instances. A real-valued vector of dimension X is shown by $\mathbb{R}^X$, and a probability simplex of dimension $X-1$ is shown by $\mathbb{P}^X$ denotes: the space of all probability distributions on a X-value random variable. To denote $\{0, 1, \ldots, X-1\}$ we use $[X]$, and $\lfloor\cdot\rceil$ denotes rounding to the nearest integer. All logarithms are natural unless written explicitly otherwise. Sigmoid function is $\sigma(\mathbf{x}) = 1/\big(1+\exp(-\mathbf{x})\big)$. Given a discrete random variable $x \in [X]$, where possible outcomes occur with probabilities $\{p_0, p_1, \ldots, p_{X-1}\}$, the entropy of x is $\mathsf{H}(x) = -\sum_{i=0}^{X-1} p_i \log(p_i)$ and the mutual information (MI) between x and another variable y is: $\mathtt{I}(x; y) = \mathsf{H}(x) - \mathsf{H}(x|y)$. $\mathbb{I}_{(\mathrm{C})}$ shows the indicator function that outputs 1 if condition C holds, and 0 otherwise.

Definitions. Let us consider a *user* owning data $\mathbf{x} \in \mathbb{R}^M$ sampled from an unknown data distribution $\mathcal{D}$. Let $\mathbf{x}$ be informative about at least two latent categorical variables (*attributes*): $y \in [Y]$ as the *target* attribute, and $s \in [S]$ as the *sensitive* attribute (see Figure 1). Let a server own a *classifier* $\mathcal{F}$ that takes $\mathbf{x}$ and outputs: $\hat{\mathbf{y}} = \mathcal{F}(\mathbf{x}) = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_Y]$, where $\hat{y}_i$ estimates the probability of $y = i$ given $\mathbf{x}$. Let $\overline{y}$ denote the predicted value for y that is decided from $\hat{\mathbf{y}}$; *e.g.,* based on a threshold function in binary classification or argmax function in multi-class classification. We assume that $\mathbf{x}$, and all intermediate computations of $\mathcal{F}$, are hidden and the user only releases $\hat{\mathbf{y}}$. Let $\hat{\mathbf{s}} = \mathcal{G}(\hat{\mathbf{y}})$ be the *attack* on the sensitive attribute s that only the server knows about. Let $\overline{s}$ denote the predicted value for s that is decided based on $\hat{\mathbf{s}}$. In sum, the following Markov chain holds: $(y, s) \longrightarrow \mathbf{x} \longrightarrow \hat{\mathbf{y}} \longrightarrow \hat{\mathbf{s}}$. Let us define the following terminology:

1. Honesty. Given a test dataset $\mathbb{D}^{test} \sim \mathcal{D}$, we define $\mathcal{F}$ as a δ^y-*honest* classifier if

$$\Pr_{(\mathbf{x},y)\sim\mathbb{D}^{test},\ \overline{y}\leftarrow\hat{\mathbf{y}}\leftarrow\mathcal{F}(\mathbf{x})}[\overline{y} = y] \geq \delta^y,$$

where $\delta^y \in [0, 1]$ is known as the classifier's test accuracy, and we call it the *honesty* of $\mathcal{F}$.

2. Curiosity. Given a test dataset $\mathbb{D}^{test} \sim \mathcal{D}$ and an attack $\mathcal{G}$, we define $\mathcal{F}$ as a δ^s-*curious* classifier if

$$\Pr_{(\mathbf{x},s)\leftarrow\mathbb{D}^{test},\ \hat{\mathbf{y}}\leftarrow\mathcal{F}(\mathbf{x}),\ \bar{s}\leftarrow\mathcal{G}(\hat{\mathbf{y}})}[\bar{s} = s] \geq \delta^s,$$

where $\delta^s \in [0, 1]$ is the attack's success rate on the test set, and we call it the *curiosity* of $\mathcal{F}$.
3. Honest-but-Curious (HBC). We define $\mathcal{F}$ as a (δ^y, δ^s)-HBC classifier if it is both δ^y-*honest* and δ^s-*curious* on the same $\mathbb{D}^{test}$.
Finally, we call $\mathcal{F}$ a *standard* classifier if $\mathcal{F}$ is trained only for achieving the best honesty, and no intended curiosity is applied.

In the following sections we show how a server, using a training dataset $\mathbb{D}^{train}$, can train an HBC classifier to establish efficient *honesty-curiosity* trade-offs, *i.e.,* (δ^y, δ^s), and analyze privacy risks and characteristics of HBC classifiers compared to standard classifiers (see Appendix E for more motivational examples).

3 BLACK-BOX VIEW: AN EFFICIENT ATTACK

To build a better intuition on our problem, we first discuss a *black-box* view where server is free in choosing the classifier, and we show the existence of an efficient attack for every task with $S < Y$.

3.1 A Convex Classifier

We start with a very basic logistic regression classifier. Let us consider the synthetic data distribution depicted in Figure 2 where each sample $\mathbf{x} \in \mathbb{R}^2$ has two attributes $y \in \{0, 1\}$ and $s \in \{0, 1\}$. When a sensitive attribute is correlated with the target attribute, the classifier's output always reveals some sensitive information (we also explore this in real-world datasets in Section 5) . The less a sensitive attribute is correlated with the target attribute, the more difficult it should be for the server to build an HBC classifier. Thus, we build the data distribution in Figure 2 such that y and s are independent, and for each attribute, there is an optimal linear classifier.

It is clear that for this dataset, we can find an optimal logistic regression classifier $\hat{y} = \sigma(w_1x_1 + w_2x_2 + b)$, with parameters $[w_1, w_2, b]$, that simulates the decision boundary of y. Such a classifier is δ^y-*honest* with $\delta^y = 1$, and at the same time it is δ^s-*curious* with $\delta^s = 1/2$; that means the classifier is honest and does not leak any sensitive information. Any effort for making this linear classifier curious to achieve $\delta^s > 1/2$ will hurt the honesty by forcing a $\delta^y < 1$. For this logistic regression classifier, it can be shown that for any HBC classifier we have $\delta^y = 1.5 - \delta^s$. For example, the optimal linear classifier for attribute s cannot have a better performance than a random guess on attribute y.

In Appendix A, we show how a logistic regression classifier can become HBC with a convex loss function, and analyze the behavior of such a classifier in detail. Specifically, we show that the trade-off that we face for a classifier with *limited* capacity (*e.g.,* logistic regression) is that: if alongside the target attribute, we also optimize for the sensitive attribute, we will only ever converge to a neighborhood of the optimum for the target attribute. We show that the size of the neighborhood is getting larger by the weight (*i.e.,* importance) we give to curiosity. While the result in Appendix A hold for a convex setting and are simplistic in nature, this analysis provides intuitions into the idea that when the attributes y and s are somehow *correlated*, the output $\hat{y}$ can better encode both tasks, and when we have *independent* attributes, we need classifiers with more capacity to cover the payoff for not converging to the optimal point of target attribute.

3.2 A Mixture of Two Classifiers

Figure 3 shows how two logistic regression classifiers, each trained separately for a corresponding attribute, can be combined such that the final output $\hat{y}$ is a mixture of the predicted values for target attribute y, and the sensitive attribute s. There are two ways to combine $z^y \in [0, 1]$ and $z^s \in [0, 1]$. Considering multipliers $\beta^y \in [0, 1]$ and $\beta^s = 1 - \beta^y$, one option is the *normal* mixture where $\hat{y} = \beta^y z^y + \beta^s z^s$, and another one is the *hard* mixture where $\hat{y} = \beta^y \lfloor z^y \rceil + \beta^s \lfloor z^s \rceil$. Since two classifiers are each optimal for the dataset in Figure 2, $\hat{y}$ in hard

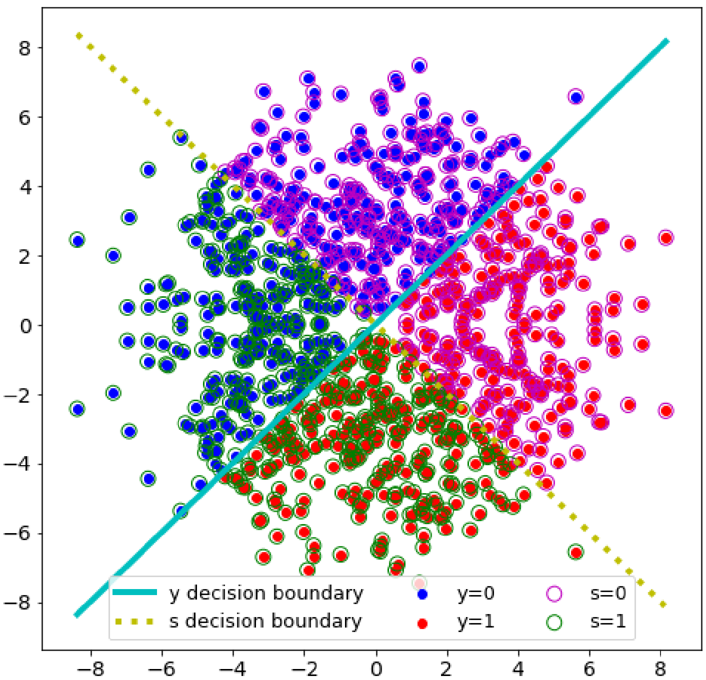

Figure 2: A two-attribute two-class data distribution on $\mathbb{R}^2$, where attributes y and s are independent, and classes in each attribute are linearly separable.

mixture can only take four values:

$$\hat{y} = \begin{cases} 0 & \text{if } y = 0, s = 0 \\ \beta^s & \text{if } y = 0, s = 1 \\ \beta^y & \text{if } y = 1, s = 0 \\ 1 & \text{if } y = 1, s = 1. \end{cases}$$

Thus, as long as we choose $\beta^y \neq 0.5$, given a $\hat{y}$, we can accurately estimate both y and s. This results in a $(\delta^y = 1, \delta^s = 1)$-HBC classifier. Notice that the classifier in Figure 3 is not a linear classifier anymore and has a larger capacity than a simple logistic regression.

The normal mixture is challenging as the possible values for $\hat{y}$ can take any value in $[0, 1]$. An idea is to define a threshold τ and divide the range of $[0, 1]$ into four sub-ranges such that:

$$\text{if} \begin{cases} \hat{y} \in [0, \tau) & \text{then } \overline{y} = 0, \overline{s} = 0 \\ \hat{y} \in [\tau, 0.5) & \text{then } \overline{y} = 0, \overline{s} = 1 \\ \hat{y} \in [0.5, 1 - \tau) & \text{then } \overline{y} = 1, \overline{s} = 0 \\ \hat{y} \in [1 - \tau, 1) & \text{then } \overline{y} = 1, \overline{s} = 1. \end{cases} \tag{1}$$

As we see in Figure 4, a normal mixture cannot guarantee to achieve the optimal $(\delta^y = 1, \delta^s = 1)$-HBC that we could obtain via a hard mixture; even in this simple case. Nevertheless, in the following sections, we will show that the idea of dividing the range $[0, 1]$ into four sub-ranges is still useful in situations where a hard mixture approach is not achievable but we can have non-linear classifiers.

Overall, in a black-box view the server can always train two separate classifiers, each with sufficiently high accuracy, and can use the hard mixture of two outputs to build the best achievable (δ^y, δ^s)-HBC classifier; no matter what type of classifier is used. Basically, the server can mostly get the same performance if it could separately run two classifiers on the data. However, we emphasize that the motivation for such a mixture classifier,

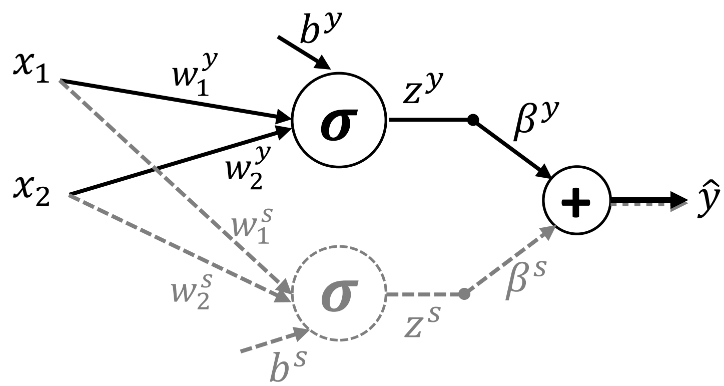

Figure 3: Two logistic regression classifiers, each trained separately: one on y and another on s. Classifiers outputs, z^y and z^s are mixed with each other such that the final output, $\hat{y}$, is a real-valued scalar informative about both attributes.

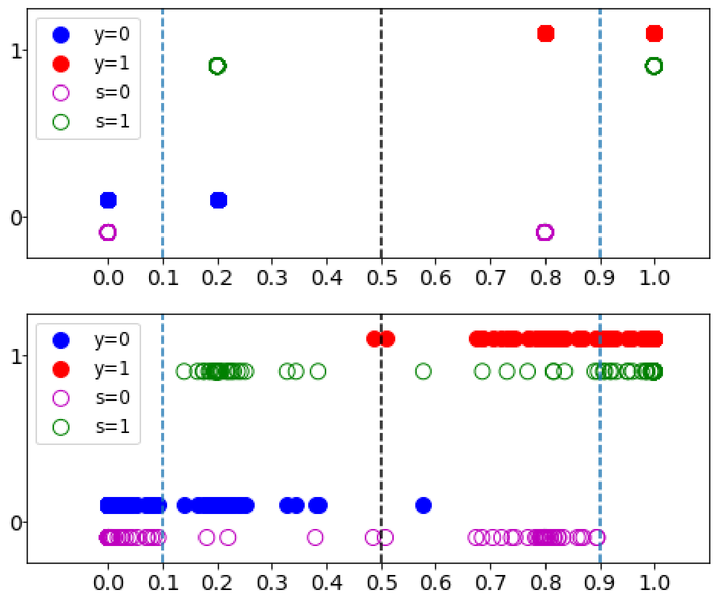

Figure 4: The outputs of classifier in Figure 3 trained on the dataset in Figure 2, using $\beta^y = .8$ and $\beta^s = .2$. (Top) the hard mixture ŷ with $(\delta^y = 1, delta^s = 1)$. (Bottom) the normal mixture ŷ with $(\delta^y = .99, delta^s = .95)$ using $\tau = .1$ in Eq. (1).

and not just simply using two separate classifiers, is that the shape of ultimate output is decided by Y. Therefore, a limitation is that such a mixture model works only if $S \leq Y$, which in general is not always the case. For instance, let $Y = 5$, $S = 3$, $\beta^y = .8$, and $\beta^s = .2$. If the hard output of the classifier trained for y is $\lfloor \mathbf{z}^y \rceil = [0, 0, 0, 1, 0]$ and the classifier trained for s outputs $\mathbf{z}^s = [0, 1, 0]$, then by observing $\hat{\mathbf{y}} = [0, 0.2, 0, 0.8, 0]$, server can estimate both y and s while looking very honest. But if $S > Y$, then server cannot easily encode the private attribute via this method, because we assume that the cardinality of the classifier's output is limited to Y as it has to look like a standard classifier. Thus, the main challenge is when $S > Y$, or more importantly, when users have a white-box view and the server is not free to choose any arbitrary architecture, *e.g.*, it has to train an of-the-shelf classifier. In the following, we focus on the white-box view in a general setting.

4 WHITE-BOX VIEW: A GENERAL SOLUTION

Theoretically, the classifier's output $\hat{\mathbf{y}}$, as a real-valued vector, can carry infinite amount of information. Thus, releasing $\hat{\mathbf{y}}$ without imposing any particular constraint can reveal any private information and even can be used to (approximately) reconstruct data $\mathbf{x}$. One can imagine a hash function that maps each $\mathbf{x}$ to a specific $\hat{\mathbf{y}}$, and consequently, by observing $\hat{\mathbf{y}}$, we can reconstruct $\mathbf{x}$ [52]. However, the complexity of real-world data, assumptions on the required honesty, white-box view, requirement of soft outputs, and other practical constraints will rule out such trivial solutions. Here, we discuss the connection between the curiosity of a classifier and the entropy of its output, and then we formulate the problem of establishing a desired trade-off for an HBC classifier $\mathcal{F}$ and its corresponding attack $\mathcal{G}$ into an information-theoretical framework.

4.1 Curiosity and Entropy

The entropy of a random variable x is the expected value of the information content of that variable; also called self-information: $\mathtt{H}(x) = \mathtt{I}(x; x)$ [33]. When we are looking for specific information in the data, then the presence of other potentially unrelated information in that data could make extracting the right information more challenging. In principle, such data would act as noise for our target task. For example, when looking for a target attribute, a trained DNN classifier takes data $\mathbf{x}$ (usually with very high entropy) and produces $\hat{\mathbf{y}} \in [Y]$ as a probability distribution over the possible outcomes with much lower-entropy compared to the original data, and although $\hat{\mathbf{y}}$ contains much less information than $\mathbf{x}$ in the sense that it has less entropy, it is considered more informative *w.r.t.* the target y. On the other hand, there is a relationship between the entropy of a classifier's output, $\mathtt{H}(\hat{\mathbf{y}}) \in [0, \log(Y)]$, and the curiosity δ^s of an attack $\mathcal{G}$. The larger $\mathtt{H}(\hat{\mathbf{y}})$, the more information is carried by $\hat{\mathbf{y}}$, thus the higher the chance to reveal information unrelated to the target task. For example, assume that $Y = 4$, $y = 1$, and y is independent of s. In the extreme case when the classifier's output is $\hat{\mathbf{y}} = [0, 1, 0, 0]$ (that means $\mathtt{H}(\hat{\mathbf{y}}) = 0$), then $\hat{\mathbf{y}}$ carries no information about s and adding any information about s would require increasing the entropy of the output $\hat{\mathbf{y}}$.

In supervised learning, the common loss function for training DNN classifiers is *cross entropy*: $\mathtt{H}_y(\hat{\mathbf{y}}) = -\sum_{i=0}^{Y-1} \mathbb{I}_{(y=i)} \log(\hat{y}_i)$; that inherently minimizes $\mathtt{H}(\hat{\mathbf{y}})$ during training time. However, since data is usually noisy at inference time, we cannot put any upper bound on $\mathtt{H}_y(\hat{\mathbf{y}})$. It is empirically shown [16] that explicitly adding an entropy minimization regularizer, alongside the cross-entropy loss, may help in keeping $\mathtt{H}(\hat{\mathbf{y}})$ low at inference time, which turns out to be useful for some applications like semi-supervised learning. But there is no way to guarantee that a classifier will always produce a minimum- or bounded-entropy output at inference time. This fact somehow serves as the main motivation of our work for encoding private attributes of the classifier's input into the classifier's output.

4.2 Regularized Attack

We introduce an attack for building an HBC classifier in situations where sensitive attribute is binary, *i.e.,* $S = 2$, and we only have access to the soft output, *i.e.,* $\hat{\mathbf{y}} \in \mathbb{P}^Y$. The idea is to enforce classifier $\mathcal{F}$ to explicitly inject sensitive information into the entropy of $\hat{\mathbf{y}}$ by *regularizing* the loss function on $\mathcal{F}$. In general, there are two properties of $\hat{\mathbf{y}}$ that we can utilize for creating an HBC classifier:

1. Argmax: as it is the usual practice, we use the index of the maximum value of $\hat{\mathbf{y}}$ to infer y; thus this helps the classifier to satisfy the honesty requirement.

2. Entropy: the entropy of $\hat{\mathbf{y}}$ can have at least two states: (i) be close to the maximum entropy, *i.e.,* $\log(Y)$, or (ii) be close to the minimum entropy, *i.e.,* 0.

We show that entropy can be used for inferring a binary sensitive attribute, while not interfering with the `argmax` of $\hat{\mathbf{y}}$ that is preserved for the target attribute y. Without loss of generality, let us assume $Y = 2$. Consider observing $\hat{y} \in \mathbb{P}$; that is equivalent to $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2] \in \mathbb{P}^2$, as $\hat{y}_1 = 1 - \hat{y}_2$. Figure 5 shows how we can take advantage of the single real-valued $\hat{y}$ to infer two attributes. For example both $\hat{\mathbf{y}} = [.95, .05]$ and $\hat{\mathbf{y}} = [.75, .25]$ have the same `argmax` but different entropies: .29 and .81, respectively.

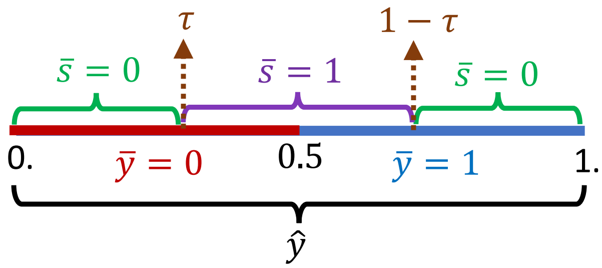

Figure 5: Observing $\hat{y} \in [0,1]$, we can predict $\overline{y}=0$ if $\hat{y} < .5$, otherwise $\overline{y}=1$. We train the classifier such that for samples with $s=0$, $\hat{y}$ gets close to the borders (0 or 1) depending on y; otherwise, for $s=1$, $\hat{y}$ gets far from the borders. Using a threshold τ, we predict binary attributes y and s from $\hat{y}$.

Training. Choosing any arbitrary classifier $\mathcal{F}$, the server can train $\mathcal{F}$ with the following loss function:

$$\begin{aligned}\mathcal{L}^b =& -\beta^y \sum_{i=0}^{Y-1} y_i \log \hat{y}_i \\ & - \beta^s \Big(\mathbb{I}_{(s=0)} \big(\sum_{i=0}^{Y-1} \hat{y}_i \log \hat{y}_i \big) - \mathbb{I}_{(s=1)} \big(\sum_{i=0}^{Y-1} \hat{y}_i \log \hat{y}_i \big) \Big),\end{aligned} \tag{2}$$

where multipliers β^y and β^s are used to control the trade-off between honesty and curiosity. In Eq. (2), in the first term, we have the cross-entropy and in the second term, we have Shannon entropy that aims to minimize the entropy of $\hat{\mathbf{y}}$ for samples of $s=0$, while maximizing the entropy of $\hat{\mathbf{y}}$ for samples of $s=1$.

At inference time, when the server observes $\hat{\mathbf{y}}$, it computes the entropy of $\hat{\mathbf{y}}$: $\mathsf{H}(\hat{\mathbf{y}}) = -\sum_{i=0}^{Y-1} \hat{y}_i \log \hat{y}_i$, and using a threshold $\tau \in [0,1]$, that in practice can be fine-tuned using a validation set at training time (we show this in Section 5), server estimates $\bar{s}$:

$$\bar{s} = \mathcal{G}(\hat{\mathbf{y}}) = \begin{cases} 0, & \text{if } \mathsf{H}(\hat{\mathbf{y}}) \leq \tau \\ 1, & \text{otherwise} \end{cases} \tag{3}$$

4.3 Parameterized Attack

In this section, we present our general solution that works for $S \geq 2$ and for both raw and soft outputs.

4.3.1 An Information Bottleneck Formulation. Remember the Markov chain $(y,s) \longrightarrow \mathbf{x} \longrightarrow \hat{\mathbf{y}} \longrightarrow \hat{\mathbf{s}}$. We assume that server is constrained on a specific family of ML models $\mathbb{F}$, *e.g.,* a specific DNN architecture that has to be as honest as a standard classifier. Server looks for a function $\mathcal{F}^* \in \mathbb{F}$ to map the users' data $\mathbf{x}$ into a vector $\hat{\mathbf{y}}$ such that $\hat{\mathbf{y}}$ is as informative about both y and s as possible.

Formally, $\mathcal{F}^*$ can be defined as the solution of the following mathematical optimization:

$$\min_{(\mathbf{x},y,s)\leftarrow\mathcal{D},\ \mathcal{F}\in\mathbb{F},\ \hat{\mathbf{y}}\leftarrow\mathcal{F}(\mathbf{x})} \big[\mathcal{I} = \beta^x \mathtt{I}(\hat{\mathbf{y}};\mathbf{x}) - \beta^y \mathtt{I}(\hat{\mathbf{y}};y) - \beta^s \mathtt{I}(\hat{\mathbf{y}};s) \big], \tag{4}$$

where β^x, β^y, and β^s are Lagrange multipliers that allow us to move along different possible local minimas and all are non-negative real-valued[1]. Eq. (4) is an extension of the *information bottleneck* (IB) formulation [65], where the optimal $\hat{\mathbf{y}}$, produced by $\mathcal{F}^*$, is decided based on its relation to the three variables, $\mathbf{x}$, y, and s. By varying the β parameters, we can explore the trade-off between compression at various rates ,*i.e.,* by minimizing $\mathtt{I}(\hat{\mathbf{y}};\mathbf{x})$, and the amount of information we aim to preserve, *i.e.,* by maximizing $\mathtt{I}(\hat{\mathbf{y}};y)$ and $\mathtt{I}(\hat{\mathbf{y}};s)$. Particularly for DNNs, it is shown that compression might help the classifier to achieve better generalization[66].

For better understanding, let us assume, for standard classifiers with $\beta^s=0$, the optimal standard solution $\mathcal{F}^*$ obtains a specific value for $\mathcal{I}$. Now, assume that the server aims to find an HBC solution, by setting $\beta^s > 0$.

[1]Mathematically speaking, we only need two Lagrange multipliers as β^y and β^s are dependent. Here we use three multipliers for the ease of presentation.

Then, in order to maintain the same value of $\mathcal{I}$ with the same mutual MI for the target attribute $\mathtt{I}(\hat{\mathbf{y}}; y)$, the term $\mathtt{I}(\hat{\mathbf{y}}; \mathbf{x})$ must be increased, because $\mathtt{I}(\hat{\mathbf{y}}; s) \geq 0$. Since for deterministic classifiers $\mathtt{I}(\hat{\mathbf{y}}; \mathbf{x}) = \mathtt{H}(\hat{\mathbf{y}})$, we conclude that when server wants to encode information about both y and s in the output $\hat{\mathbf{y}}$, then the output's entropy $\mathtt{H}(\hat{\mathbf{y}})$ of $\mathcal{F}^*$ must be higher than if server was only encoding information about y; which shows that our formulation in Eq. (4) is consistent with our motivation for exploiting the capacity of $\mathtt{H}(\hat{\mathbf{y}})$. In Section 5, we provide more intuition on this through some experimental results.

4.3.2 Variational Estimation. Now, we analyze a server that is computationally bounded and has only access to a sample of the true population (*i.e.,* a training dataset) and wants to solve Eq. (4) to create a (near-optimal) HBC classifier $\mathcal{F}$. We have

$$\mathcal{I} = \beta^x \mathtt{I}(\hat{\mathbf{y}}, \mathbf{x}) - \beta^y \mathtt{I}(\hat{\mathbf{y}}; y) - \beta^s \mathtt{I}(\hat{\mathbf{y}}; s) =$$
$$\beta^x \mathtt{H}(\hat{\mathbf{y}}) - \beta^x \mathtt{H}(\hat{\mathbf{y}}|\mathbf{x}) - \beta^y \mathtt{H}(y) + \beta^y \mathtt{H}(y|\hat{\mathbf{y}}) - \beta^s \mathtt{H}(s) + \beta^s \mathtt{H}(s|\hat{\mathbf{y}}).$$

Since for a fixed training dataset $\mathtt{H}(y)$ and $\mathtt{H}(s)$ are constant during the optimization and for deterministic functions $\mathtt{H}(\hat{\mathbf{y}}|\mathbf{x}) = 0$, we can reduce Eq. (4) as

$$\min_{(\mathbf{x},y,s)\leftarrow\mathcal{D},\ \mathcal{F}\in\mathbb{F},\ \hat{\mathbf{y}}\leftarrow\mathcal{F}(\mathbf{x})} \left[\mathcal{H} = \beta^x \mathtt{H}(\hat{\mathbf{y}}) + \beta^y \mathtt{H}(y|\hat{\mathbf{y}}) + \beta^s \mathtt{H}(s|\hat{\mathbf{y}})\right]. \tag{5}$$

Eq. (5) can be interpreted as an optimization problem that aims to minimize the entropy of $\hat{\mathbf{y}}$ subject to encoding as much information as possible about y and s into $\hat{\mathbf{y}}$. Thus, the optimization seeks for a function $\mathcal{F}^*$ to produce a low-entropy $\hat{\mathbf{y}}$ such that $\hat{\mathbf{y}}$ is only informative about y and s and no information about anything else. Multipliers β^y and β^s specify how y and s can compete with each other for the remaining capacity in the entropy of $\hat{\mathbf{y}}$; that is challenging, particularly, when y and s are independent.

Different constraints on the server can lead to different optimal models. As we observed, in a black-box view with $S \leq Y$ and arbitrary $\mathbb{F}$, a solution is achieved by training two separate classifiers with a cross-entropy loss function and an entropy minimization regularizer [16]. First, using stochastic gradient decent (SGD), we train classifier $\mathcal{F}^y$ by setting $\beta^y = 1$ and $\beta^s = 0$ in Eq. (5). Second, we train classifier $\mathcal{F}^s$ by setting $\beta^y = 0$ and $\beta^s = 1$. Finally, we build $\mathcal{F} = \beta^y \lfloor \mathcal{F}^y \rceil + \beta^s \lfloor \mathcal{F}^s \rceil$ as the desired HBC classifier for any choice of $\beta^y \in [0, 1]$ and $\beta^s = 1 - \beta^y$. Notice that the desired value for β^x can be chosen through a cross-validation process.

Thus, let us focus on the white-box view with a constrained $\mathbb{F}$, *e.g.,* where the server is required to train a normal architecture or an off-the-shelf classifier. Here, we use the cross-entropy loss function for the attribute y and train the specified classifier $\mathcal{F}$ using SGD. However, besides this cross-entropy loss, we also need to look for another loss function for the attribute s. Thus, we need a method to simulate such a loss function for s.

Let $p_{s|\hat{\mathbf{y}}}$ denote the true, but unknown, probability distribution of s given $\hat{\mathbf{y}}$, and $q_{s|\hat{\mathbf{y}}}$ denote an approximation of $p_{s|\hat{\mathbf{y}}}$. Considering the cross entropy between these two distributions $\mathtt{H}(p_{s|\hat{\mathbf{y}}}, q_{s|\hat{\mathbf{y}}})$, it is known [35] that

$$\mathtt{H}(s|\hat{\mathbf{y}}) = -\sum_{i=0}^{S-1} p_{s|\hat{\mathbf{y}}} \log(p_{s|\hat{\mathbf{y}}}) \leq \mathtt{H}(p_{s|\hat{\mathbf{y}}}, q_{s|\hat{\mathbf{y}}}) = -\sum_{i=0}^{S-1} p_{s|\hat{\mathbf{y}}} \log(q_{s|\hat{\mathbf{y}}}). \tag{6}$$

This inequality tell us that the cross entropy between the unknown distribution, *i.e.,* $p_{s|\hat{\mathbf{y}}}$, and any estimation of it, *i.e.,* $q_{s|\hat{\mathbf{y}}}$, is an upper-bound on $\mathtt{H}(s|\hat{\mathbf{y}})$; and the equality holds when $q_{s|\hat{\mathbf{y}}} = p_{s|\hat{\mathbf{y}}}$. Thus, if we find a useful model for $q_{s|\hat{\mathbf{y}}}$, then the problem of minimizing $\mathtt{H}(s|\hat{\mathbf{y}})$ can be solved through minimization of $\mathtt{H}(p_{s|\hat{\mathbf{y}}}, q_{s|\hat{\mathbf{y}}})$.

In practice, parameterized models such as neural networks, are suitable candidates for $q_{s|\hat{\mathbf{y}}}$ [35, 50]. For N i.i.d. samples of pairs $\left\{(\hat{\mathbf{y}}^1, s^1), \ldots, (\hat{\mathbf{y}}^N, s^N)\right\}$ that represent $p_{s|\hat{\mathbf{y}}}$ and are generated via our current classifier $\mathcal{F}$ on a dataset $\mathbb{D}^{train} \sim \mathcal{D}$, we can estimate $\mathtt{H}(p_{s|\hat{\mathbf{y}}}, q_{s|\hat{\mathbf{y}}})$ using the empirical cross-entropy as

$$\hat{\mathtt{H}}^N(p_{s|\hat{\mathbf{y}}}, q_{s|\hat{\mathbf{y}}}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=0}^{S-1} \mathbb{I}_{(s=i)} \log\left(q_{s|\hat{\mathbf{y}}}(\hat{\mathbf{y}}^n)\right). \tag{7}$$

Therefore, after initializing a *parameterized* model for $q_{s|\hat{\mathbf{y}}}$ to estimate $\mathtt{H}(s|\hat{\mathbf{y}})$ in Eq. (5), we run optimization

$$\hat{\mathtt{H}}(p_{s|\hat{\mathbf{y}}}, q_{s|\hat{\mathbf{y}}}) = \min_{q_{s|\hat{\mathbf{y}}}} \hat{\mathtt{H}}^N(p_{s|\hat{\mathbf{y}}}, q_{s|\hat{\mathbf{y}}}), \tag{8}$$

where we iteratively sample N pairs $(s^i, \hat{\mathbf{y}}^i)$, compute Eq. (7), and update the model $q_{s|\hat{\mathbf{y}}}$; using SGD.

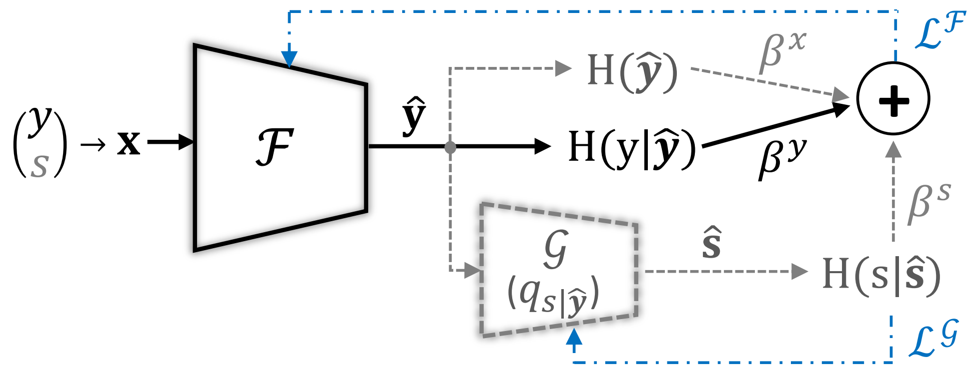

Figure 6: The server iteratively trains the chosen classifier $\mathcal{F}$ and attack $\mathcal{G}$ that is a *parameterized* model. While $\mathcal{G}$ is optimized only based on $\mathsf{H}(s|\hat{s})$, classifier $\mathcal{F}$ is optimized based on a weighted combination of $\mathsf{H}(\hat{\mathbf{y}})$, $\mathsf{H}(y|\hat{\mathbf{y}})$ and $\mathsf{H}(s|\hat{\mathbf{y}})$ in Eq. (9).

Training. The server will use this additional model $q_{s|\hat{\mathbf{y}}}$ for s, alongside the cross entropy loss function for y, to solve Eq. (5) for finding the optimal $\mathcal{F}$. Considering $q_{s|\hat{\mathbf{y}}}$ as the attack $\mathcal{G}$ and setting our desired β multipliers, the variational approximation of our general optimization problem in Eq. (5) is written as

$$\min_{(\mathbf{x},y,s)\leftarrow\mathcal{D},\ \mathcal{F}\in\mathbb{F},\ \hat{\mathbf{y}}\leftarrow\mathcal{F}(\mathbf{x}),\hat{\mathbf{s}}=\mathcal{G}(\hat{\mathbf{y}})} \left[\mathcal{H} = \beta^x \mathsf{H}(\hat{\mathbf{y}}) + \beta^y \mathsf{H}(y|\hat{\mathbf{y}}) + \beta^s \mathsf{H}(s|\hat{\mathbf{y}})\right], \tag{9}$$

where the minimization is performed over both parameterized models $\mathcal{F}$ and $\mathcal{G}$. A schematic view of this approach is depicted in Figure 6. Using a training dataset $(\mathbf{x}, y, s) \in \mathbb{D}^{train} \sim \mathcal{D}$ and in an iterative process, both classifier $\mathcal{F}$, with loss function $\mathcal{L}^{\mathcal{F}} = \mathcal{H}$ and attack $\mathcal{G}$, with loss function $\mathcal{L}^{\mathcal{G}} = \mathsf{H}(s|\hat{\mathbf{y}})$, are simultaneously trained. Algorithm 1, in Appendix B, shows the details of our proposed training method.

5 EVALUATION

We present results in several tasks from two real-world datasets and discuss the honesty-curiosity trade-off for different attributes.

5.1 Experimental Setup

5.1.1 Settings. We define a *task* as training a specific classifier $\mathcal{F}$ on a training dataset including samples with two attributes, $y \in [Y]$ and $s \in [S]$, and evaluating $\mathcal{F}$ on a test dataset by measuring honesty δ^y and curiosity δ^s via the attack $\mathcal{G}$. We have two types of attacks: (i) *regularized*, as explained in Section 4.2, and (ii) *parameterized*, as explained in Section 4.3. For each task there are 3 different scenarios: (1) Std: where $\mathcal{F}$ is a standard classifier without intended curiosity, (2) RawHBC: where $\mathcal{F}$ is trained to be HBC and $\mathcal{G}$ has access to raw outputs $\hat{\mathbf{y}} \in \mathbb{R}^Y$, and (3) SoftHBC: where $\mathcal{F}$ is HBC but $\mathcal{G}$ has access only to soft outputs $\hat{\mathbf{y}} \in \mathbb{P}^Y$. We run more than 200 experiments, each five times, and report mean and standard deviation. For each experiment in SoftHBC, $\tau \in [0, 1]$ in Eq. (3) is chosen based on the validation set and is used to evaluate the result on the test set.

5.1.2 CelebA Dataset [32]. This is a dataset including more than 200K celebrity face images, each with 40 binary attributes, *e.g.*, the ‘Smiling’ attribute with values 0:*Non-Smile* or 1:*Smile*. We choose attributes that are almost balanced, meaning that there are at least 30% and at most 70% samples for that attribute with value 1. Our chosen attributes are: Attractive, BlackHair, BlondHair, BrownHair, HeavyMakeup, Male, MouthOpen, Smiling, and WavyHair. CelebA is already split into separate training, validation, and test sets. We use the resampled images of size 64×64. We elaborate more on CelebA and show some samples of this dataset in Appendix F.1.

Table 1: (A) The characteristics of four attributes vs. Smiling attribute in the training set of CelebA dataset. The honesty (δ^y) and curiosity (δ^s) of (B) standard classifiers when releasing either raw or soft outputs, as well as our proposed HBC classifiers trained for (C) regularized attacks and (D) parameterized attacks. Here, $Y = S = 2$ and $\beta^x = 0$. Values are in percentage (%).

(A) For each attribute: the empirical joint probability and mutual information (MI) with Smiling, besides accuracy of the standard classifier (δ^y)

MouthOpen				**Male**				**HeavyMakeup**				**WavyHair**			
	Non-Smile	Smile	sum		Non-Smile	Smile	sum		Non-Smile	Smile	sum		Non-Smile	Smile	sum
0	.386	.122	.508	0	.292	.375	.667	0	.305	.184	.489	0	.322	.166	.488
1	.103	.389	.492	1	.197	.136	.333	1	.226	.28.5	.511	1	.303	.209	.512
sum	.489	.510		sum	.489	.511		sum	.531	.469		sum	.625	.375	
	MI: .231	δ^y: 93.42 ± .07			MI: .015	δ^y: 97.67 ± .04			MI: .024	δ^y: 88.89 ± .19			MI: .003	δ^y: 77.29 ± .55	

	Setting	(β^y, β^s)	s: **MouthOpen**		s: **Male**		s: **HeavyMakeup**		s: **WavyHair**	
			δ^y	δ^s	δ^y	δ^s	δ^y	δ^s	δ^y	δ^s
			(B) Overlearning [61]							
y: **Smiling**	Std raw	(1., *0.*)	92.15 ± .04	*79.56 ± .32*	92.15 ± .04	*68.20 ± .03*	92.15 ± .04	*60.96 ± .10*	92.15 ± .04	*57.93 ± .21*
	Std soft			*79.22 ± .14*		*68.20 ± .00*		*60.53 ± .11*		*57.90 ± .00*
			(C) Regularized Attack							
	SoftHBC	(.7,.3)	91.90 ± .01	84.73 ± .50	91.78 ± .15	90.14 ± .37	92.03 ± .12	80.79 ± .47	92.11 ± .15	68.06 ± .72
		(.5, .5)	91.83 ± .13	89.08 ± .04	91.65 ± .10	94.20 ± .18	91.87 ± .03	85.27 ± .15	91.98 ± .18	72.36 ± .83
		(.3, .7)	91.75 ± .10	91.22 ± .25	91.60 ± .11	96.02 ± .09	91.58 ± .10	87.09 ± .43	91.73 ± .07	73.52 ± .77
			(D) Parameterized Attack							
	RawHBC	(.7,.3)	91.84 ± .07	93.40 ± .09	92.36 ± .02	97.21 ± .10	92.12 ± .09	88.63 ± .04	92.17 ± .04	76.74 ± .37
	SoftHBC	(.7,.3)	91.49 ± .44	85.94 ± .76	91.72 ± .17	79.39 ± 2.4	91.84 ± .10	84.45 ± .15	92.12 ± .09	57.90 ± .00
		(.5,.5)	90.20 ± 1.1	88.91 ± .83	90.17 ± .66	92.73 ± .76	91.49 ± .11	86.19 ± .18	92.20 ± .12	61.51 ± .77
		(.3,.7)	82.55 ± .27	93.15 ± .07	67.65 ± 6.4	97.41 ± .06	70.75 ± 4.3	88.62 ± .52	89.27 ± 1.9	67.57 ± 3.7

5.1.3 UTKFace Dataset [79]. This is a dataset including 23,705 face images and annotated with attributes gender (male or female), race (White, Black, Asian, Indian, or others), and age (0-116). We use the resampled images of size 64×64, and randomly split UTKFace into subsets of sizes 18964 (80%), 4741 (20%) for training and test set, respectively. A subset of 1896 images (10%) from the training set is randomly chosen as the validation set for training. See Appendix F.2 for the details of tasks we define on UTKFace, and some samples.

5.1.4 Architectures. For $\mathcal{F}$, we use a DNN architecture similar to the original paper of UTKFace dataset [79] that includes 4 convolutional layers and 2 fully-connected layers with about 250K trainable parameters. For $\mathcal{G}$, we use a very simple 3-layer fully-connected classifier with about 2K to 4K trainable parameters; depending on Y. The implementation details for $\mathcal{F}$ and $\mathcal{G}$ are presented in Appendix G. For all experiments, we use a batch size of 100 images, and Adam optimizer [27] with learning rate .001. After fixing β multipliers, we run training for 50 epochs, and choose models of the epoch that both $\mathcal{F}$ and $\mathcal{G}$ achieve the best trade-off for both y and s (based on β^y and β^s) on the validation set, respectively. That is, the models that give us the largest $\beta^y\delta^y + \beta^s\delta^s$ on the validation set during training. Notice that, the fine-tuning is a task at training time, and a server with enough data and computational power can find near-optimal values for (β^y, β^s), as we do here using the validation set. In the following, all reported values for honesty δ^y and curiosity δ^s of HBC classifiers are the accuracy of the final $\mathcal{F}$ and $\mathcal{G}$ on the test set. Finally, in all Std settings, values in *italic* show the effect of overlearning [61], that is the accuracy of a parameterized $\mathcal{G}$ in inferring a sensitive attribute from a standard classifier.

5.2 UTKFace: Gender vs. Race

As the first result, we present a simple result on UTKFace. We set Gender as the target attribute and Race (White, Non-White) as the sensitive attribute. The training set includes 52% Male and 48% Female, where 42% of samples are labeled as White and 58% as Non-White. Figure 7 shows the ROC curves for this experiment. In the top-left plot, the standard classifier achieves 0.97 AUC, whereas in the bottom-left plot classifier is HBC but it still achieves a considerable 0.94 AUC. On the other hand, the standard classifier in the top-right plot is not informative about Race and it basically is as good as a random guess. But, the HBC classifier in the bottom-right can achieve 0.89 AUC on predicting Race via the regularized attack.

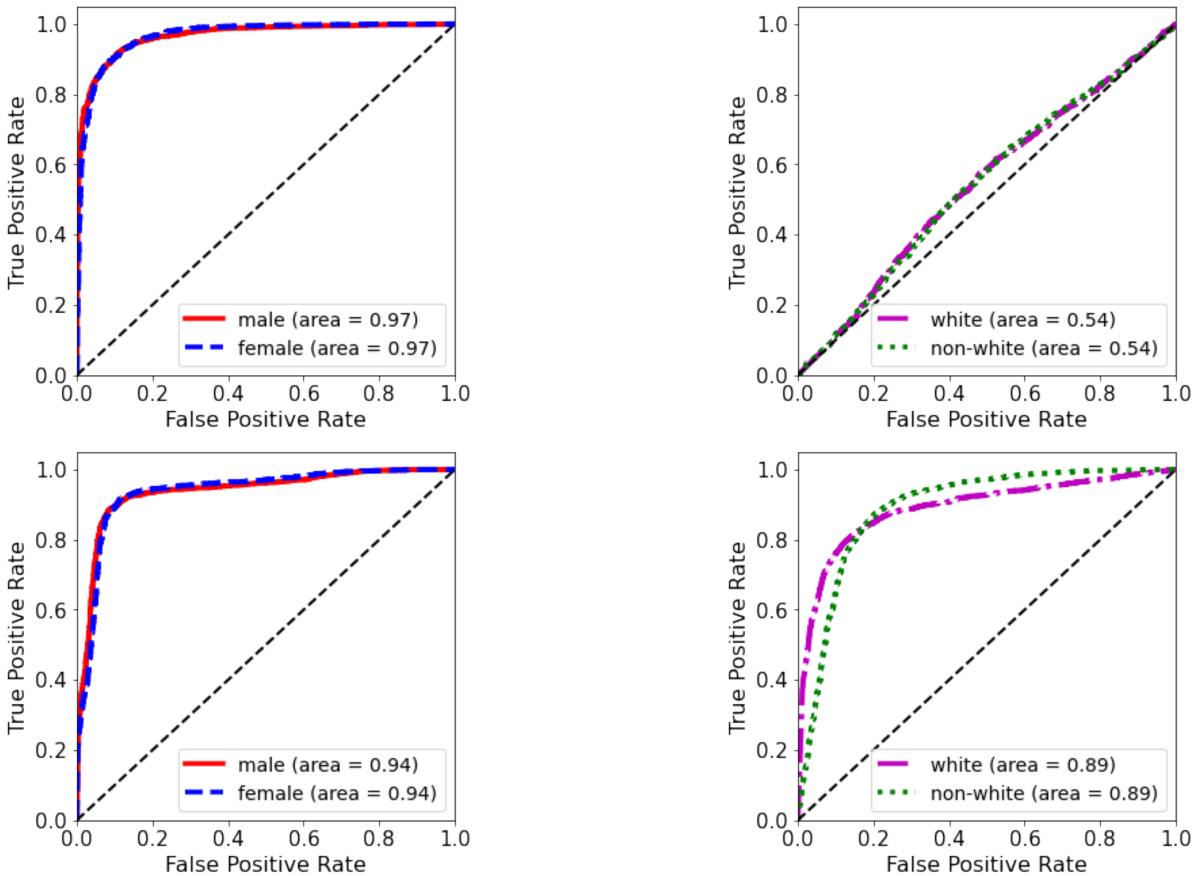

Figure 7: The ROC curve of (top) a standard classifier and (bottom) an HBC classifier trained by the regularized attack in SoftHBC**. The target attribute is Gender and the sensitive attribute is Race (in UTKFace), and** (β^y, β^s) **in Eq.** (2) **is** $(.5, .5)$**.**

5.3 CelebA: Smiling vs. Other Attributes

Here, we consider a smile detection task on CelebA where the target attribute y is set to Smiling and the sensitive attribute s is set to one of MouthOpen, Male, HeavyMakeup, or WavyHair. To better understand the results, Table 1 (A) shows the characteristics of these attributes compared to Smiling attribute in the training dataset. As expected, MouthOpen is the most correlated one (MI is .231 bits) while WavyHair is the least correlated attribute (MI is .003 bits) to Smiling. We also show the test accuracy of the $\mathcal{F}$ as a standard classifier trained for each attribute. We see that Male is the easiest attribute (97% accuracy) and WavyHair is the most difficult one (77% accuracy). In the following, we see how these factors, *correlation* and *easiness*, affect the performance of HBC classifiers. Table 1 (B, C, and D) show the performance of HBC classifiers for different attacks and trade-offs based on β^y and β^s. Our findings are:

1. The outputs of a standard binary classifier, either raw or soft, do not reveal information about sensitive attributes more than what one could already infer from the knowledge about underlying data distribution. Thus, while overlearning [61] has shown serious problems when the adversary has access to a subset of internal representations, here we observe that when one only has access to the output of a binary classifier, then overlearning is not a major problem (we show this in other settings as well). On the other hand, HBC classifiers can effectively learn to encode the sensitive attribute in a (more restricted than the internal representation) single-valued output.

2. In regularized attacks in Table 1 (B), with a very small loss in δ^y (less than 1%), we can get δ^s very close to the accuracy that we could have achieved if we could run a separate classifier for the sensitive attribute. In parameterized attacks, it is easier to encode sensitive information into the raw output than the soft output. The attacks in RawHBC are highly successful in the curiosity, in all four cases in Table 1 (C), almost without any damage to the honesty. On the other hand, in SoftHBC it is harder to establish an efficient trade-off between honesty and curiosity. Since $\text{Softmax}(\mathbf{x}) = \text{Softmax}(\mathbf{x} + a)$ for all $a \in \mathbb{R}$, there are infinitely many vectors in $\mathbb{R}^Y$ (in RawHBC) that can be mapped into the same vector in $\mathbb{P}^Y$ (in SoftHBC). However, what one can learn about the sensitive attribute in SoftHBC is still much more than Std. In the following, we will see that when $Y > 2$, parameterized attack in SoftHBC is also very successful.

3. The easiness of the sensitive attribute plays an important role. For example, in face image processing, gender classification is in general an easier task than detecting heavy makeup (as it might be easier for human beings as well). Therefore, while MI between Smiling and HeavyMakeup is larger than Smiling and Male, the curiosity in inferring Male attribute is more successful than HeavyMakeup. Moreover, while (due to MI) δ^s of MouthOpen is about 11% more than Male in Std, and in contrast to overlearning attack, the correlation is not that important in HBC settings compared to the easiness, as we see that for Male all attacks are as successful as MouthOpen. For the same reason, for attribute WavyHair it is more difficult to achieve high curiosity as it is not an easy task. It is worth noting that, for difficult attributes we may be able to improve the curiosity by optimizing $\mathcal{F}$ for that specific attribute, *e.g.,* through neural architecture search. But for fair comparisons, we use the same DNN for all experiments and we leave the optimization of DNN to future studies.

5.4 CelebA: HairColor vs. Other Attributes

Moving beyond binary classifiers, in Table 2 we present a use-case of three-class classifier (Y=3) for the target attribute of HairColor, where there are 40% samples of BlackHair, 34% BrownHair, and 26% BlondHair. We consider three sensitive attributes with different degrees of easiness: (A) Male, (B) Smiling, and (C) Attractive. While in Std setting, attacks on overlearning [61] are not very successful (even when releasing the raw outputs), our parameterized attacks, in RawHBC, are very successful without any meaningful damage to the honesty of classifier. In SoftHBC, while it is again harder to train a parameterized attack as successful as RawHBC, we do find successful trade-offs if we fine-tune (β^y, β^s); particularly for the regularized attack.

5.5 UTKFace: Sensitive Attributes with S>2

To evaluate tasks with $S > 2$, we provide results of several experiments performed on UTKFace in Tables 3 and Table 4 (also some complementary results in Appendix C, Tables 7 and Table 8). In each experiment, we set one of Gender, Age, or Race, as the target attribute and another one as the sensitive attribute, and compare the achieved δ^y and δ^s. See Appendix F.2 for the details of how we created labels for different values of Y and S. Our findings are:

1. An HBC classifier can be as honest as a standard classifier while also achieving a considerable curiosity. For all RawHBC cases, the δ^y of an HBC classifier is very close to δ^y of a corresponding classifier in Std. Moreover, we see that in some situations, making a classifier HBC even helps in achieving a better generalization and consequently getting a slightly better honesty; which is very important as an HBC classifier can look as honest as possible (we elaborate more on the cause of this observation in Appendix D).

2. When having access to raw outputs, the attack is highly successful in all tasks, and in many cases, we can achieve similar accuracy to a situation where we could train $\mathcal{F}$ for that specific sensitive attribute. For example in Table 3 for $S = 3$ and $Y > 2$, we can achieve about 83% curiosity in inferring the Race attribute from a classifier trained for Age attribute. When looking at Table 7 where Race is the target attribute, we see that the best accuracy a standard classifier can achieve for race classification is about 85%.

3. In SoftHBC, it is more challenging to achieve a high curiosity via a parameterized attack, unless we sacrifice more honesty. Particularly for tasks with $S > Y$ where the sensitive attribute is more granular than the target attribute. Also, while we observed successful regularized attacks for SoftHBC in Table 1 (C), but regularized

Table 2: Results where the target attribute is HairColor and the sensitive attribute is (A) Male, (B) Smiling, or (C) Attractive. Here Y=3, S=2, and β^x=0. Values are in percentage (%).

Setting	Attack	(β^y, β^s)	δ^y	δ^s
(A) Male (class distribution 0:66%, 1:34%)				
Std	raw [61]	(1., *.0*)	92.94 ± .58	*75.86 ± .03*
RawHBC	Parameterized	(.7, .3)	92.85 ± .42	97.27 ± .09
SoftHBC	Parameterized	(.7, .3)	92.12 ± .27	94.23 ± 1.4
		(.5, .5)	91.98 ± .12	96.87 ± .13
		(.3, .7)	67.63 ± .25	97.52 ± .08
	Regularized	(.7, .3)	92.79 ± .22	95.11 ± .22
		(.5, .5)	92.78 ± .28	96.68 ± .12
		(.3, .7)	92.96 ± .33	97.02 ± .04
(B) Smiling (class distribution 0:49%, 1:51%)				
Std	raw [61]	(1., *.0*)	92.94 ± .58	*56.12 ± .21*
RawHBC	Parameterized	(.7, .3)	92.56 ± .46	91.79 ± .12
SoftHBC	Parameterized	(.7, .3)	92.96 ± .44	88.59 ± 1.8
		(.5, .5)	91.27 ± .35	89.18 ± .06
		(.3, .7)	88.05 ± 1.3	91.07 ± .51
	Regularized	(.7, .3)	92.80 ± .28	86.22 ± .38
		(.5, .5)	92.52 ± .43	90.67 ± .15
		(.3, .7)	92.31 ± .32	91.59 ± .07
(C) Attractive (class distribution 0:40%, 1:60%)				
Std	raw [61]	(1., *.0*)	92.94 ± .58	*60.73 ± .21*
RawHBC	Parameterized	(.7, .3)	92.81 ± .16	76.97 ± .07
SoftHBC	Parameterized	(.7, .3)	93.03 ± .33	71.54 ± .60
		(.5, .5)	92.62 ± .64	76.18 ± .12
		(.3, .7)	92.66 ± .33	75.57 ± .37
	Regularized	(.7, .3)	91.95 ± .35	69.06 ± .39
		(.5, .5)	92.33 ± .44	74.38 ± .13
		(.3, .7)	91.09 ± .46	75.70 ± .27

attacks cannot be applied to tasks with $S > 2$. Yet, even in this case of having only access to soft outputs, the curiosity is much higher than what can be learned from the raw output of a standard classifier (via overlearning attack). Although the curiosity in RawHBC is more successful than SoftHBC, the differences between these two are getting smaller as the size of output Y is getting larger.

4. Attacks are highly successful when $S \leq Y$, as there is more capacity in the released output. However, the attack is successful in scenarios when $S > Y$ as well. The most challenging case is where $Y = 2$ and when we only have access to the soft outputs, because in these tasks we only release one value(*i.e.,* $\hat{y}_1 = 1 - \hat{y}_2$). Moreover, we see that in Table 4 for SoftHBC with $S = Y = 2$, regularized attacks achieve much better trade-offs than parameterized attacks.

5.6 Entropy Minimization with $\beta^x > 0$

We examine the entropy minimization (*i.e.,* compression) of the classifier's output and its effect on the achieved trade-off for honesty and curiosity. Table 5 compares the results of different values chosen for β^x in Eq. (9) (also see Figure 6). An important observation is that compression is mostly helpful to the honesty. This is expected, due to our discussion in Section 4.3, and findings in previous related work [66, 68]. Moreover, we see that compression

Table 3: The honesty (δ^y) and curiosity (δ^s) of an HBC classifier trained via parameterized attack, where the target attribute (y) is Age and the sensitive attribute (s) is Race. Values are in percentage (%)

			$S=2$		$S=3$		$S=4$		$S=5$	
		(β^y,β^s)	δ^y	δ^s	δ^y	δ^s	δ^y	δ^s	δ^y	δ^s
$Y=2$	NC	(1.,*0.*)	83.03 ± .33	*62.44 ± .97*	83.03 ± .33	*52.34 ± 1.4*	83.03 ± .33	*45.74 ± .13*	83.03 ± .33	*44.67 ± .17*
	RawHBC	(.7,.3)	83.60 ± .22	86.77 ± .11	83.22 ± .27	80.06 ± .88	83.10 ± .18	74.12 ± .33	83.07 ± .13	72.52 ± .34
	SoftHBC	(.7,.3)	82.87 ± .21	69.89 ± .40	82.88 ± .27	63.28 ± .30	81.67 ± .54	55.01 ± .57	81.94 ± .81	53.84 ± .93
		(.5,.5)	68.86 ± .93	84.75 ± 1.1	64.60 ± 1.0	82.88 ± .56	72.28 ± .65	62.87 ± .29	65.71 ± .86	71.54 ± 1.6
$Y=3$	NC	(1.,*0.*)	81.09 ± .37	*65.99 ± .40*	81.09 ± .37	*56.37 ± .47*	81.09 ± .37	*48.28 ± .49*	81.09 ± .37	*46.67 ± .41*
	RawHBC	(.7,.3)	81.67 ± .33	86.35 ± .65	81.35 ± .28	83.42 ± .29	81.23 ± .29	76.62 ± .56	81.30 ± .28	76.24 ± .59
	SoftHBC	(.7,.3)	81.19 ± .02	79.07 ± .15	80.76 ± .36	72.90 ± .23	80.10 ± .32	66.40 ± 1.1	80.29 ± .07	67.74 ± .30
		(.5,.5)	78.17 ± .38	86.17 ± .20	69.56 ± .75	80.14 ± .30	69.03 ± .60	76.63 ± .19	68.54 ± .98	76.69 ± .72
$Y=4$	NC	(1.,*0.*)	68.59 ± .24	*66.93 ± .39*	68.59 ± .24	*58.02 ± .55*	68.59 ± .24	*49.23 ± .53*	68.59 ± .24	*41.13 ± .15*
	RawHBC	(.7,.3)	68.59 ± .27	86.91 ± .27	68.59 ± .49	84.17 ± .26	68.29 ± .26	79.46 ± .16	68.40 ± .50	78.79 ± .13
	SoftHBC	(.7,.3)	68.15 ± .42	78.10 ± .32	67.45 ± .25	74.70 ± .43	66.30 ± .22	69.01 ± 1.3	65.70 ± .51	69.37 ± .28
		(.5,.5)	64.48 ± .50	86.68 ± .11	62.69 ± .19	84.11 ± .51	58.95 ± .77	77.48 ± .53	58.19 ± .57	76.89 ± .99
$Y=5$	NC	(1.,*0.*)	62.24 ± .61	*67.00 ± .32*	62.24 ± .61	*58.69 ± .44*	67.00 ± .32	*50.37 ± .35*	67.00 ± .32	*49.22 ± .62*
	RawHBC	(.7,.3)	62.72 ± .20	86.63 ± .05	62.46 ± .49	83.79 ± .16	61.84 ± .77	78.53 ± .80	62.40 ± .31	78.28 ± .20
	SoftHBC	(.7,.3)	62.08 ± .14	79.41 ± .75	61.35 ± .58	74.57 ± .91	60.90 ± .39	69.32 ± .49	60.70 ± .52	69.56 ± 1.2
		(.5,.5)	58.53 ± .85	86.88 ± .34	56.63 ± .79	84.06 ± .09	53.27 ± .14	77.45 ± .11	54.61 ± .20	77.30 ± 1.0

Table 4: The honesty (δ^y) and curiosity (δ^s) of an HBC classifier trained via different attacks, where the target attribute (y) is Race and the sensitive attribute (s) is Gender. Values are in percentage (%).

				$Y=2$		$Y=3$		$Y=4$		$Y=5$	
	Setting	Attack	(β^y,β^s)	δ^y	δ^s	δ^y	δ^s	δ^y	δ^s	δ^y	δ^s
$S=2$	Std	raw [61]	(1.,*0.*)	87.67 ± .49	*56.11 ± .44*	85.50 ± .13	*56.04 ± 2.4*	81.15 ± .23	*58.91 ± .74*	80.69 ± .23	*61.06 ± 2.0*
	RawHBC	Parameterized	(.7, .3)	88.10 ± .22	89.36 ± .03	85.77 ± .30	89.14 ± .21	82.05 ± .14	89.19 ± .33	81.43 ± .34	89.00 ± .42
	SoftHBC	Parameterized	(.7, .3)	87.43 ± .09	51.66 ± .00	85.71 ± .16	83.51 ± 1.1	81.10 ± .23	83.60 ± .22	80.46 ± .26	88.46 ± .18
			(.5, .5)	83.67 ± .27	80.17 ± 2.1	82.89 ± .59	85.69 ± .54	70.36 ± 1.5	86.05 ± 1.6	77.40 ± .35	88.36 ± .45
		Regularized	(.7, .3)	87.30 ± .29	75.29 ± .89	84.89 ± .20	77.53 ± .33	80.64 ± .31	76.29 ± .45	80.24 ± .09	78.01 ± .62
			(.5, .5)	86.61 ± .15	83.23 ± .27	83.89 ± .44	86.61 ± .61	79.03 ± .21	86.29 ± .32	78.61 ± .65	85.97 ± .28

Table 5: The effect of entropy minimization on the UTKFace dataset for the target attribute Age and the sensitive attribute Race where $Y = S = 3$. We also show the average of the entropy of classifier's output by $\widetilde{\mathsf{H}}(\hat{\mathbf{y}})$ in bits.

	(β^y,β^s)		$\beta^x=.0$			$\beta^x=.2$			$\beta^x=.4$			$\beta^x=.8$	
		δ^y	δ^s	$\widetilde{\mathsf{H}}(\hat{\mathbf{y}})$	δ^y	δ^s	$\widetilde{\mathsf{H}}(\hat{\mathbf{y}})$	δ^y	δ^s	$\widetilde{\mathsf{H}}(\hat{\mathbf{y}})$	δ^y	δ^s	$\widetilde{\mathsf{H}}(\hat{\mathbf{y}})$
Std	(1., *.0*)	81.43±.37	*58.14±.06*	.48±.05	81.04±.09	*56.13±.14*	.45±.06	80.90±.27	*58.60±.07*	.40±.05	81.15±.41	*55.94±.26*	.32±.02
RawHBC	(.7, .3)	81.32±.31	82.97±.27	.63±.05	81.59±.42	82.90±.24	.45±.02	81.62±.37	81.98±.77	.36±.02	81.61±.31	81.40±.41	.28±.03
	(.5, .5)	80.23±.42	84.82±.35	.76±.03	80.60±.14	84.60±.32	.54±.03	80.95±.41	83.86±.54	.39±.01	81.26±.22	83.92±.32	.26±.01
SoftHBC	(.7, .3)	81.04±.26	73.34±.29	.72±.02	81.26±.28	69.77±.50	.51±.02	81.01±.55	64.54±2.5	.37±.04	81.01±.46	56.28±.51	.27±.05
	(.5, .5)	69.12±.91	80.81±.97	1.05±.01	76.20±.29	76.27±.37	.76±.01	80.13±.15	74.01±.46	.58±.01	80.59±.16	70.28±.15	.33±.02

is more effective in improving the honesty of the classifier in situations that we assign more weight to the curiosity of the classifier.

Although Table 5 shows that large compression hurts curiosity more, this is another trade-off that a server can utilize to make the HBC classifier less suspicious. It is important that the average entropy of classifier's output, shown by $\widetilde{\mathsf{H}}(\hat{\mathbf{y}})$, is directly related to the curiosity weight β^s. The more we focus on making a classifier curious, the larger will be the entropy of the output. This confirms our main hypothesis that the output's entropy can be utilized for encoding sensitive attributes. In Figure 8, we plot the histogram of normalized observations of the output's entropy for the setting of Y=S=3 for five scenarios in Table 5. We see that the output of an HBC classifier tends to have larger entropy than a standard classifier. Moreover, large entropies, *i.e.,* more than 1 bit, are more common in SoftHBC than RawHBC. A reason for this is that the capacity of soft outputs is smaller than raw outputs; hence, the classifier tends to take more advantage of the existing capacity. Interestingly, when we use

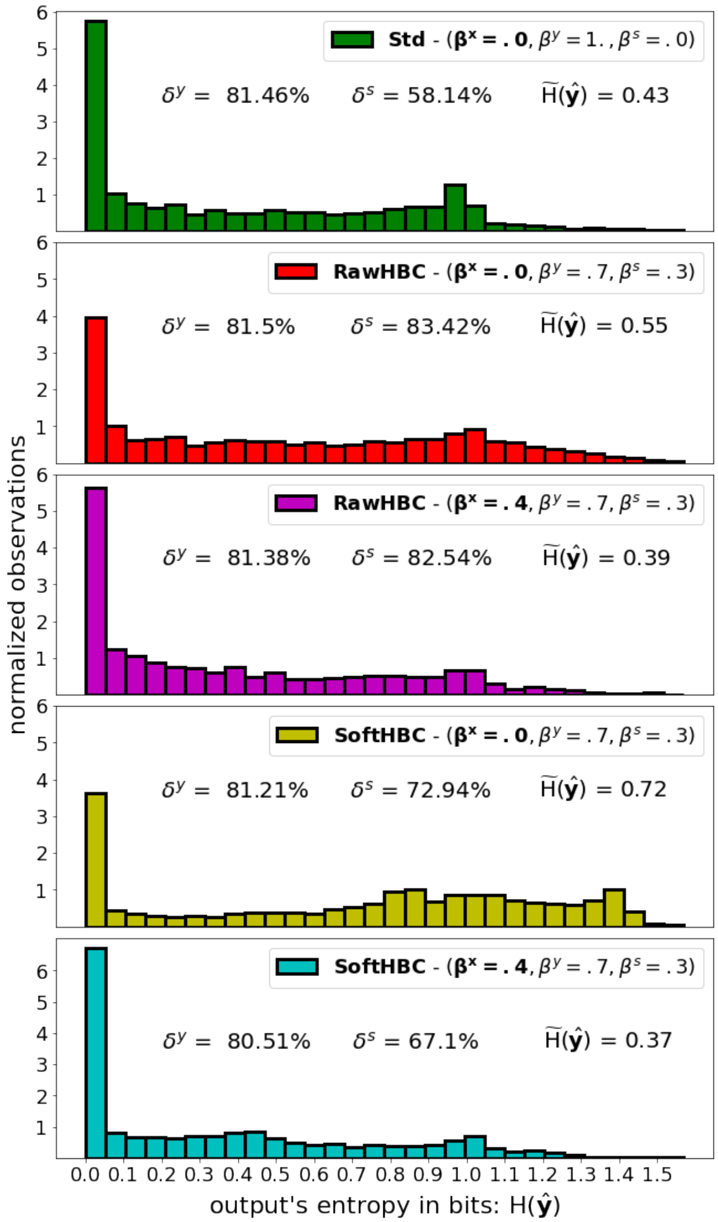

Figure 8: The normalized density histogram (*i.e.,* number of observations) of the output entropies of classifier with and without compression. Here, y is Age and s is Race with $Y = S = 3$ on UTKFace. From top to bottom, classifier is trained (1) standard, (2, 3) RawHBC **where we release the raw outputs, and (4, 5)** SoftHBC **where we release the soft outputs. For (2) and (4) we set $\beta^x = .0$, and for (3) and (5) we set $\beta^x = .4$.**

the entropy minimization with $\beta^x > 0$, then we observe that the entropy distribution for an HBC classifier has a smaller tail (compared to $\beta^x = 0$) and thus can become even less suspicious than the standard classifier when we release the raw output. Finally, in releasing the soft outputs, it is a bit more challenging to keep the average entropy low.

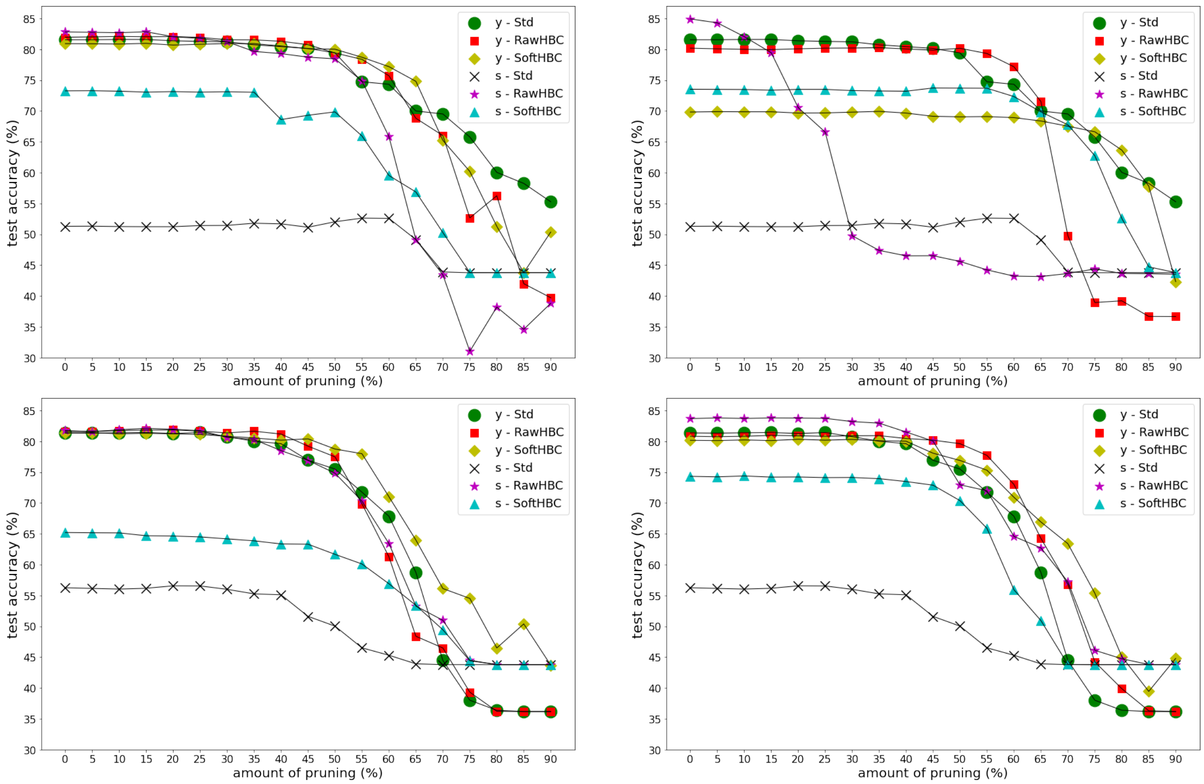

Figure 9: The effect of pruning, with L1 norm, where the classifier $\mathcal{F}$ when y is Age, s Race, and $Y = S = 3$. (Top-Left) $\beta^x, \beta^y, \beta^s = (0, .7, .3)$. (Top-Right) we set $\beta^x, \beta^y, \beta^s = (0, .5, .5)$.(Bottom-Left) $\beta^x, \beta^y, \beta^s = (.4, .7, .3)$. (Bottom-Right) we set $\beta^x, \beta^y, \beta^s = (.4, .5, .5)$.

5.7 Pruning HBC Classifiers

As DNN classifiers are mostly overparameterized, a hypothesis might be that HBC classifiers utilize the extra capacity of DNNs for extracting patterns that correspond to the sensitive attribute. Thus, one could say that reducing a DNN's capacity, using a pruning technique [19], might make it more difficult for a classifier to be curious. For example, a user who does not trust a server can take the classifier $\mathcal{F}$ and perform some pruning technique, before making inferences, hoping that pruning will not damage the honesty but will reduce curiosity. Figure 9 shows the honesty-curiosity trade-off for different amounts of pruning. As the pruning technique, we use *L1-Unstructured* implemented in [51], where parameters with the lowest L1-norm are set to zero at inference time (*i.e.,* no re-training).

1. Comparing the top plots (where $\beta^x = 0$) with bottom plots (where $\beta^x = 0.4$) in Figure 9, we see that classifiers without compression show more tolerance to a large amount of pruning (>60%) than classifiers with compression. This might be due to the additional constraint that we put on classifiers with compression.

2. By pruning less than 50% of the parameters, there is no significant drop in the honesty in both standard and HBC classifiers. However, for the curiosity, HBC classifiers show different behaviors in different settings. When there is no compression ($\beta^x = 0$), the curiosity of RawHBC shows faster and much larger drops, compared to SoftHBC. When there is compression ($\beta^x = 0.4$), then drops are not different across these settings.

3. If we prune more than 50% of the parameters, the drop in accuracy for both the target and sensitive attributes are significant. However, SoftHBC settings interestingly show better performances than RawHBC, for both target and sensitive attributes. While RawHBC has usually shown better performance in previous sections, in this specific case SoftHBC performs interestingly very well.

Table 6: An HBC teacher can train an HBC student via KD. The target attribute is Smiling in CelebA and the setting is SoftHBC **with** $(\beta^x, \beta^y, \beta^s) = (.0, .5, .5)$**.**

s	classifier	Parameterized δ^y	Parameterized δ^s	Rgularized δ^y	Rgularized δ^s
Mouth Open	Teacher	90.04 ± .61	89.14 ± .68	91.61 ± .24	89.01 ± .34
	Student	90.30 ± .47	88.07 ± .65	91.29 ± .11	83.69 ± .38
Male	Teacher	90.17 ± .66	92.73 ± .76	91.57 ± .16	94.11 ± .21
	Student	90.20 ± .51	91.62 ± .73	91.01 ± .19	85.71 ± .46
Heavy Makeup	Teacher	90.73 ± 1.1	85.17 ± .97	91.56 ± .21	85.58 ± .27
	Student	91.00 ± .43	82.62 ± .49	91.23 ± .15	81.05 ± .15
Wavy Hair	Teacher	91.99 ± .21	61.05 ± 1.8	91.86 ± .08	71.12 ± 1.2
	Student	91.78 ± .17	59.72 ± 1.1	91.60 ± .15	68.61 ± .96

In sum, although Figure 9 shows that pruning can damage curiosity more than honesty in settings without compression. We cannot guarantee that pruning at inference time is an effective defense against HBC models, as adding compression constraint will help the server to make HBC models more tolerable to pruning, and with the amount of pruning up to 50% the curiosity still is high.

5.8 Attack on Knowledge Distillation

Transferring knowledge from a *teacher* classifier, trained on a large labeled dataset, to a *student* classifier, that has access only to a (small) unlabeled dataset, is know as "knowledge distillation" (KD) [18, 20]. Since KD allows to keep sensitive data private, it has found some applications in privacy-preserving ML [47, 70]. Let us consider a user that owns a private unlabeled dataset and wants to train a classifier on this dataset, and a server that provides a teacher classifier trained on a large labeled dataset. A common technique in KD is to force the student to mimic the teacher's behavior by minimizing the KL-divergence between the teacher's soft outputs, $\hat{\mathbf{y}}^{Teacher}$, and student's soft outputs, $\hat{\mathbf{y}}^{Student}$ [20]:

$$\mathcal{L}^{KL} = \sum_{i=1}^{Y} \hat{\mathbf{y}}_i^{Teacher} \log \left(\hat{\mathbf{y}}_i^{Teacher} / \hat{\mathbf{y}}_i^{Student} \right). \tag{10}$$

We show that if the teacher is HBC, then the student trained using $\mathcal{L}^{KL}$ can also become HBC. We run experiments, similar to Section 5.3, by assigning 80% of the training set (with both labels) to the server, and 20% of the training set (without any labels) to the user. At the server side, we train an HBC teacher via both regularized and parameterized attacks, and then at the user side, we use the trained teacher classifier to train a student only via $\mathcal{L}^{KL}$ and the user's unlabeled dataset. We also set the student's DNN to be half the size of the teacher's in terms of the number of trainable parameters in each layer. We set $(\beta^y, \beta^s) = (.5, .5)$ and consider the SoftHBC setting, because $\mathcal{L}^{KL}$ is based on mimicking the teacher's soft outputs.

Table 6 shows that KD works well for the honesty in both cases; showing that an HBC teacher can look very honest in transferring knowledge of the target attribute. For the curiosity, the parameterized attacks transfer the knowledge very well and usually better than regularized attacks. A reason might be that for regularized attacks the chosen threshold τ for the teacher classifier is not the best choice for replicating the attack in its students. Though, for WavyHair as an uncorrelated and difficult attribute, the regularized attack achieves a better result. Overall, this capability of transferring curiosity shows another risk of HBC classifiers provided by semi-trusted servers, especially that such a teacher-student approach has shown successful applications in semi-supervised learning [22, 53, 63, 77]. We note that, there are several aspects, such as the teacher-student data ratio and distribution shift, or different choices of the loss function, that may either improve or mitigate this attack. We leave further investigation into these for future studies.

6 DISCUSSIONS AND RELATED WORK

We discuss related work and potential proactive defenses against the vulnerabilities highlighted in this paper.

6.1 Proactive Investigation

The right to information privacy is long known as "the right to select what personal information about me is known to what people" [74]. The surge of applying ML to (almost) all tasks in our everyday lives has brought attention to the ethical aspects of ML [25]. To reconstruct the users' face images from their recordings of speech [45, 73] raised the concern of "tying the identity to biology" and categorizing people into gender or sexual orientation groups that they do not fit well [21]. The estimation of ethnicity [69] or detecting sexual orientation [72] from facial images has risen concerns on misusing such ML models by central authorities that seek to determine people of minority groups. It is shown [44] that a widely used healthcare model exhibits significant racial bias, causing Black patients to receive less medical care than others. The costs and potential risks associated with large-scale language models, such as discriminatory biases, are discussed in [4] and the research community is encouraged to consider the impacts of the ever-increasing size of DNNs beyond just the model's accuracy for a target task.

In this paper, we showed another major concern regarding DNNs that enables attacks on the users' privacy; even when users might think it is safe to only release a very narrow result of their private data. An important concern on the potential misuse of ML models is that unlike the discovery of software vulnerabilities that can be quickly patched, it is very difficult to propose effective defenses against harmful consequences of ML models [57]. It is suggested [26] that tech regulators should become "proactive", rather than being "reactive", and design controlled, confidential, and automated experiments with black-box access to ML services. While service providers may argue that ML models are proprietary resources, it is not unreasonable to allow appropriate regulators to have controlled and black-box access for regulatory purposes. For instance, a regulator can check the curiosity of ML classifiers provided by cloud APIs via a test set that includes samples with sensitive attributes.

6.2 Attacks in ML

It is shown [60] that ML models can leak detailed sensitive information about their training datasets in both white-box and black-box views. Since DNNs tend to learn as many generic low-level features as they can, and some of these features are inherently useful in inferring more than one attribute, then DNNs trained for seemingly non-sensitive attributes can implicitly learn other potentially sensitive attributes [13, 61]. While information-theoretical approaches [41, 46, 71] are proposed for training DNNs such that they do not leak sensitive attributes, it is shown that the empirical implementation of these theoretical approaches cannot effectively eliminate this risk [61]. A technique based on transfer learning is proposed in [61] to "re-purpose" a classifier trained for target attribute into a model for classifying a different attribute. However, the re-purposing of a classifier is different from building an HBC classifier as the former does not aim to look honest and the privacy violation is due to the further use of a classifier for other purposes without the consent of the training data owner.

In model extraction (*a.k.a.* stealing) attack [67], an adversary aims to build a copy of a black-box ML model, without having any prior knowledge about the model's parameters or training data and just by having access to the soft predictions provided by the model. This attack is evaluated by two objectives [23]: *test accuracy*, which measures the correctness of predictions made by the stolen model, and *fidelity*, which measures the similarity in predictions (even if it is wrong) between the stolen and the original model. Model extraction has shown the richness of a classifier's output in reconstructing the classifier itself, but in our work we show how this richness can be used for encoding sensitive attributes at inference time.

ML enables unprecedented applications, *e.g.,* automated medical diagnosis [11, 17]. As personal data,*e.g.,* medical images, are highly sensitive, privacy-preserving learning, such as federated learning [36], is proposed to train DNN classifiers on distributed data. Users who own sensitive data usually participate in training a DNN for a specified target task. Although differential privacy [10] can protect a model from memorizing its training data [1, 43], the threat model introduced in this paper is different from the commonly studied setting in property [37] or membership [54, 58] inference attacks where classifier is trained on a dataset including multiple users and the server is curious about inferring a sensitive property about users or the presence or absence of a target user in the input dataset. We consider a threat to the privacy of a single user at inference time when the server observes only outputs.

7 CONCLUSION

We introduced, and systematically studied, a major vulnerability in high-capacity HBC classifiers that are built by semi-trusted servers, where classifiers can secretly encode unintended sensitive information about their input data into their public target outputs. We translated this problem into an information-theoretical framework and proposed empirical methods that can efficiently implement such an attack on the privacy of users. We analyzed several properties of classifier outputs and specifically showed that the entropy of the outputs can represent the curiosity of the model up to a certain extent. Our results show that, even when classifier outputs are very restricted in their form, they are still rich enough to carry information about more than one attribute. Furthermore, we showed that this capability can even be transferred to other classifiers that are trained based on such an HBC classifier. Finally, while we show that even soft outputs of a classifier can be exploited for encoding sensitive information, our results suggest that it will be safer for a user to release soft outputs than raw outputs.

We suggest the following open directions for further exploration. First, to search for rigorous techniques that can help in distinguishing standard and HBC classifiers, which is in the same direction of research in understanding and explaining deep neural networks. Second, a limitation of our proposed attacks is that the sensitive attribute has to be known at training time. We suggest extending the proposed methods, or designing new methods, for encoding more than one sensitive attribute in the classifier's output, which, in general, will be more challenging, but not impossible. Third, to investigate the implication and effects of employing an HBC model in collaborative learning and multi-party ML applications, where some parties might not be fully trusted.

ACKNOWLEDGMENT

This work was funded by the European Research Council (ERC) through Starting Grant BEACON (no. 677854) and by the UK EPSRC (grant no. EP/T023600/1) within the CHIST-ERA program. Anastasia thanks JPMorgan Chase & Co for the funding received through the J.P. Morgan A.I. Research Award 2019. Views or opinions expressed herein are solely those of the authors listed.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning With Differential Privacy. In *ACM SIGSAC Conference on Computer and Communications Security (CCS).*

[2] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J Kusner, and Adrià Gascón. 2019. QUOTIENT: two-party secure neural network training and prediction. In *ACM SIGSAC Conference on Computer and Communications Security (CCS).*

[3] Donald Beaver. 1991. Perfect Privacy for Two-Party Protocols. In *DIMACS Workshop on Distributed Computing and Cryptography*, Vol. 2.

[4] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the Dangers of Stochastic Parrots: Can Language Models be Too Big. In *Conference on Fairness, Accountability, and Transparency (FAccT).*

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 8 (2013).

[6] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning Attacks against Support Vector Machines. In *International Conference on Machine Learning (ICML).*

[7] Rich Caruana. 1997. Multitask Learning. *Machine Learning* 28, 1 (1997).

[8] Benny Chor and Eyal Kushilevitz. 1991. A Zero-One Law for Boolean Privacy. *SIAM Journal on Discrete Mathematics* 4, 1 (1991).

[9] Emiliano De Cristofaro. 2020. An Overview of Privacy in Machine Learning. *arXiv:2005.08679* (2020).

[10] Cynthia Dwork, Aaron Roth, et al. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014).

[11] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. 2017. Dermatologist-Level Classification of Skin Cancer with Deep Neural Networks. *Springer Nature* 542, 7639 (2017).

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning (ICML).*

[13] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *ACM SIGSAC Conference on Computer and Communications Security (CCS).*

[14] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Springer Annual Cryptology Conference.*

[15] Oded Goldreich. 2009. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press.

[16] Yves Grandvalet and Yoshua Bengio. 2005. Semi-supervised Learning by Entropy Minimization. In *Advances in Neural Information Processing Systems (NIPS)*.

[17] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. 2016. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *Journal of American Medical Association* 316, 22 (2016).

[18] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. 2016. Cross Modal Distillation for Supervision Transfer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[19] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR)*.

[20] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Workshop on Deep Learning and Representation Learning*.

[21] Matthew Hutson. 2021. Who Should Stop Unethical A.I.? *The Newyorker Annals of Technology* (2021).

[22] Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondrej Chum. 2019. Label Propagation for Deep Semi-supervised Learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[23] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High Accuracy and High Fidelity Extraction of Neural Networks. In *USENIX Security Symposium*.

[24] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. 2018. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. In *IEEE Symposium on Security and Privacy (S&P)*.

[25] Michael Kearns and Aaron Roth. 2019. *The Ethical Algorithm: The Science of Socially Aware Algorithm Design*. Oxford University Press.

[26] Michael Kearns and Aaron Roth. 2020. Ethical Algorithm Design Should Guide Technology Regulation. *Brookings Institution's Artificial Intelligence and Emerging Technology Initiative* (2020).

[27] Diederik P Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

[28] Eyal Kushilevitz, Silvio Micali, and Rafail Ostrovsky. 1994. Reducibility and Completeness in Multi-Party Private Computations. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*.

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Springer Nature* 521, 7553 (2015).

[30] Tyler Lee and Anthony Ndirango. 2019. Generalization in Multitask Deep Neural Classifiers: A Statistical Physics Approach. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[31] Yugeng Liu, Rui Wen, Xinlei He, Ahmed Salem, Zhikun Zhang, Michael Backes, Emiliano De Cristofaro, Mario Fritz, and Yang Zhang. 2021. ML-Doctor: Holistic Risk Assessment of Inference Attacks Against Machine Learning Models. *arXiv:2102.02551* (2021).

[32] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *International Conference on Computer Vision (ICCV)*.

[33] David J.C. MacKay. 2003. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.

[34] Charles T Marx, Richard Lanas Phillips, Sorelle A Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. 2019. Disentangling Influence: Using Disentangled Representations to Audit Model Predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[35] David McAllester and Karl Stratos. 2020. Formal Limitations on the Measurement of Mutual Information. In *Conference on Artificial Intelligence and Statistics (AISTATS)*.

[36] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Conference on Artificial Intelligence and Statistics (AISTAT)*.

[37] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting Unintended Feature Leakage in Collaborative Learning. In *IEEE Symposium on Security and Privacy (S&P)*.

[38] Fatemehsadat Mirshghallah, Mohammadkazem Taram, Praneeth Vepakomma, Abhishek Singh, Ramesh Raskar, and Hadi Esmaeilzadeh. 2020. Privacy in Deep Learning: A survey. *arXiv:2004.12254* (2020).

[39] Fan Mo, Anastasia Borovykh, Mohammad Malekzadeh, Hamed Haddadi, and Soteris Demetriou. 2020. Layer-wise Characterization of Latent Information Leakage in Federated Learning. *arXiv:2010.08762* (2020).

[40] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. DarkNetz: Towards Model Privacy at the Edge using Trusted Execution Environments. In *International Conference on Mobile Systems, Applications, and Services (MobiSys)*.

[41] Daniel Moyer, Shuyang Gao, Rob Brekelmans, Greg Ver Steeg, and Aram Galstyan. 2018. Invariant Representations without Adversarial Training. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[42] Kevin P Murphy. 2021. *Probabilistic Machine Learning: An Introduction*. MIT Press.

[43] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2018. Machine Learning with Membership Privacy Using Adversarial Regularization. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[44] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. 2019. Dissecting Racial Bias in an Algorithm Used to Manage the Health of Populations. *Science* 366, 6464 (2019).

[45] Tae-Hyun Oh, Tali Dekel, Changil Kim, Inbar Mosseri, William T Freeman, Michael Rubinstein, and Wojciech Matusik. 2019. Speech2Face: Learning the Face Behind a Voice. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[46] Seyed Ali Osia, Ali Shahin Shamsabadi, Sina Sajadmanesh, Ali Taheri, Kleomenis Katevas, Hamid R Rabiee, Nicholas D Lane, and Hamed Haddadi. 2020. A Hybrid Deep Learning Architecture for Privacy-Preserving Mobile Analytics. *IEEE Internet of Things Journal* 7, 5

(2020).
[47] Nicolas Papernot, Martin Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. 2016. Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data. In *International Conference on Learning Representations (ICLR)*.
[48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS)*.
[49] Andrew Paverd, Andrew Martin, and Ian Brown. 2014. Modelling and Automatically Analysing Privacy Properties for Honest-but-Curious Adversaries. *University of Oxford Technical Report* (2014).
[50] Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. 2019. On Variational Bounds of Mutual Information. In *International Conference on Machine Learning (ICML)*.
[51] PyTorch. Accessed: 2021-05-01. Pruning Library. https://github.com/pytorch/pytorch/blob/master/torch/nn/utils/prune.py. (Accessed: 2021-05-01).
[52] Adityanarayanan Radhakrishnan, Mikhail Belkin, and Caroline Uhler. 2020. Overparameterized Neural Networks Implement Associative Memory. *Proceedings of the National Academy of Sciences* 117, 44 (2020).
[53] Scott E. Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. 2015. Training Deep Neural Networks on Noisy Labels with Bootstrapping. In *International Conference on Learning Representations (ICLR)*.
[54] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. 2018. Ml-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. In *Network and Distributed Systems Security Symposium (NDSS)*.
[55] Andrew M Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D Tracey, and David D Cox. 2019. On the Information Bottleneck Theory of Deep Learning. *Journal of Statistical Mechanics: Theory and Experiment* 2019, 12 (2019).
[56] Jürgen Schmidhuber. 2015. Deep Learning in Neural Networks: An Overview. *Elsevier Neural Networks* 61 (2015).
[57] Toby Shevlane and Allan Dafoe. 2020. The Offense-Defense Balance of Scientific Knowledge: Does Publishing AI Research Reduce Misuse?. In *AAAI/ACM Conference on AI, Ethics, and Society*.
[58] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *IEEE Symposium on Security and Privacy (S&P)*.
[59] Ravid Shwartz-Ziv and Naftali Tishby. 2017. Opening the Black Box of Deep Neural Networks via Information. *arXiv:1703.00810* (2017).
[60] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. 2017. Machine Learning Models That Remember Too Much. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.
[61] Congzheng Song and Vitaly Shmatikov. 2020. Overlearning Reveals Sensitive Attributes. In *International Conference on Learning Representations (ICLR)*.
[62] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing Properties of Neural Networks. In *International Conference on Learning Representations (ICLR)*.
[63] Antti Tarvainen and Harri Valpola. 2017. Mean Teachers are Better Role Models: Weight-averaged Consistency Targets Improve Semi-supervised Deep Learning Results. In *Advances in Neural Information Processing Systems (NIPS)*.
[64] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2017. Distributed Deep Neural Networks over the Cloud, the Edge and End Devices. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE.
[65] Naftali Tishby, Fernando C. Pereira, and William Bialek. 2000. The Information Bottleneck Method. In *Annual Allerton Conference on Communication, Control and Computing*.
[66] Naftali Tishby and Noga Zaslavsky. 2015. Deep Learning and the Information Bottleneck Principle. In *IEEE Information Theory Workshop (ITW)*.
[67] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models Via Prediction APIs. In *USENIX Security Symposium*.
[68] Matias Vera, Pablo Piantanida, and Leonardo Rey Vega. 2018. The Role of The Information Bottleneck in Representation Learning. In *IEEE International Symposium on Information Theory (ISIT)*.
[69] Cunrui Wang, Qingling Zhang, Wanquan Liu, Yu Liu, and Lixin Miao. 2019. Expression of Concern: Facial Feature Discovery for Ethnicity Recognition. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9, 1 (2019).
[70] Ji Wang, Weidong Bao, Lichao Sun, Xiaomin Zhu, Bokai Cao, and S Yu Philip. 2019. Private Model Compression via Knowledge Distillation. In *AAAI Conference on Artificial Intelligence*.
[71] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S Yu. 2018. Not Just Privacy: Improving Performance of Private Deep Learning in Mobile Cloud. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*.
[72] Yilun Wang and Michal Kosinski. 2018. Deep Neural Networks are More Accurate than Humans at Detecting Sexual Orientation from Facial Images. *Journal of personality and social psychology* 114, 2 (2018).
[73] Yandong Wen, Bhiksha Raj, and Rita Singh. 2019. Face Reconstruction from Voice using Generative Adversarial Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.
[74] Alan F Westin. 1968. Privacy and Freedom. *Washington and Lee Law Review* 25, 1 (1968).
[75] Jacob Whitehill, Gwen Littlewort, Ian Fasel, Marian Bartlett, and Javier Movellan. 2009. Toward Practical Smile Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31, 11 (2009).
[76] Ashia Wilson. 2018. *Lyapunov Arguments in Pptimization*. Ph.D. Dissertation. UC Berkeley.

[77] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. 2020. Self-training with Noisy Student Improves ImageNet Classification. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
[78] Chao Zhang, Dacheng Tao, Tao Hu, and Bingchen Liu. 2020. Generalization Bounds of Multitask Learning From Perspective of Vector-Valued Function Learning. *IEEE Transactions on Neural Networks and Learning Systems* (2020).
[79] Zhifei Zhang, Yang Song, and Hairong Qi. 2017. Age Progression/Regression by Conditional Adversarial Autoencoder. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

APPENDIX

A THE CONVEX USE CASE

In this section, we first present an example of an HBC classifier with a convex loss function, then we analyze the behavior of such a classifier.

A.1 An Example of Convex Loss

Let $\mathbf{x} = [x_0, x_1, \ldots, x_{M-1}, 1]$ denote the data sample with target attribute $y \in \{0, 1\}$, and sensitive attribute $s \in \{0, 1\}$, and for each $i \in [M]$ we have $x_i \in \mathbb{R}$. Let us consider a binary logistic regression classifier $\mathcal{F}$ that is parameterized by $\Theta \in \mathbb{R}^{M+1}$:

$$\hat{y} = \mathcal{F}(\mathbf{x}) = \sigma(\Theta^\top \mathbf{x}) = \frac{1}{1 + \exp(-\Theta^\top \mathbf{x})}.$$

To make $\mathcal{F}$ a honest-but-curious classifier, the server can choose multipliers $\beta^y \in [0, 1]$ and $\beta^s \in [0, 1]$ such that $\beta^y + \beta^s = 1$ and build the loss function

$$\begin{aligned}
\mathcal{L}^{\mathcal{F}} = &\ \ \beta^y \mathcal{L}^y + \beta^s \mathcal{L}^s \\
= &- \beta^y \big(y \log \hat{y} + (1 - y) \log (1 - \hat{y})\big) \\
&- \beta^s \big(s \log \hat{y} + (1 - s) \log(1 - \hat{y})\big) \\
= &- (\beta^y y + \beta^s s) \log \hat{y} \\
&- \big(\beta^y (1 - y) + \beta^s (1 - s)\big) \log(1 - \hat{y}) \\
= &- (\beta^y y + \beta^s s) \log \hat{y} \\
&- \big((\beta^y + \beta^s) - (\beta^y y + \beta^s s)\big) \log(1 - \hat{y}) \\
= &- (\beta^y y + \beta^s s) \log \hat{y} \\
&- \big(1 - (\beta^y y + \beta^s s)\big) \log(1 - \hat{y}) \\
= &- z \log \hat{y} - \big(1 - z\big) \log(1 - \hat{y})
\end{aligned} \tag{11}$$

where $z = \beta^y y + \beta^s s$. We know that:

(1) If function $\mathcal{A}$ is convex and $\alpha \geq 0$, then $\alpha\mathcal{A}$ is convex.
(2) If two functions $\mathcal{A}_1$ and $\mathcal{A}_2$ are convex, then $\mathcal{A}_1 + \mathcal{A}_2$ is convex.
(3) Functions $-\log \hat{y} = -\log \big(\sigma(\Theta^\top \mathbf{x})\big)$ and $-\log (1 - \hat{y}) = -\log \big(1 - \sigma(\Theta^\top \mathbf{x})\big)$, are convex *w.r.t.* the parameters of a logistic regression model [42].

Thus, considering these three facts, $\mathcal{L}^{\mathcal{F}}$ in Eq. (11) is convex.

A.2 Analysis of Convex Loss

Now, we analyze the situations in which loss functions for training HBC classifiers $\mathcal{F}$, parameterized by Θ, are convex. Let the loss function for these situations be

$$\begin{aligned}
\mathcal{L}^{\mathcal{F}} &= \beta^y \mathcal{L}^y \big(\mathcal{F}(\mathbf{x}; \Theta), y\big) + \beta^s \mathcal{L}^s \big(\mathcal{G}(\mathcal{F}(\mathbf{x}; \Theta)), s\big) \\
&= \beta^y \mathcal{L}^y(\Theta) + \beta^s \mathcal{L}^s(\Theta),
\end{aligned}$$

where we assume $\mathcal{G}$ is a fixed attack, and in the second line we just simplified the equation to focus on the trainable parameters Θ. Let Θ^y_* and Θ^s_* denote the optimal parameters that minimize $\mathcal{L}^y$ and $\mathcal{L}^s$, respectively.

Assume that, at each iteration t of training, loss functions $\mathcal{L}^{\mathrm{y}}$ and $\mathcal{L}^{\mathrm{s}}$ are μ^{y}- and μ^{s}-strongly convex functions with respect to Θ_t, respectively. Considering the gradient descent dynamics in continuous time [76],

$$d\Theta_t = -\nabla_\Theta \mathcal{L}^{\mathcal{F}} dt,$$

we can define the Lyapunov function $V_t = \frac{1}{2}||\Theta_t - \Theta_*^y||_2^2$. Observe that by strong convexity of $\mathcal{L}^{\mathrm{y}}$, the optimum of this Lyapunov function is Θ_*^y and it is unique. Now, we can analyze the dynamics of this Lyapunov function to better understand the convergence of the gradient descent dynamics when the loss function is set to be optimizing both for y and s.

First, let us consider $\beta^s = 0$, then

$$\begin{aligned} dV_t &= (\Theta_t - \Theta_*^y)^\top d\Theta_t = -(\Theta_t - \Theta_*^y)^\top \nabla_\Theta \mathcal{L}^{\mathcal{F}} dt \\ &= -\beta^y (\Theta_t - \Theta_*^y)^\top \nabla_\Theta \mathcal{L}^y(\Theta_t) dt \\ &= -\beta^y (\Theta_t - \Theta_*^y)^\top \big(\nabla_\Theta \mathcal{L}^{\mathrm{y}}(\Theta_t) - \nabla_\Theta \mathcal{L}^{\mathrm{y}}(\Theta_*^y)\big) dt \\ &\leq -\beta^y \mu^{\mathrm{y}} ||\Theta_t - \Theta_*^y||_2^2 dt, \end{aligned} \tag{12}$$

where in the second line we use the gradient descent dynamics, in the third line we use the fact that $\nabla_\Theta \mathcal{L}^y(\Theta_*^y) = 0$ (remember that we assume Θ_*^y is the set of parameters that is optimal for $\mathcal{L}^{\mathrm{y}}$), and in the last line we use the strong convexity of the function. By integrating both sides of inequality in (12) we obtain

$$V_t \leq e^{-t\beta^y \mu^{\mathrm{y}}} V_0, \tag{13}$$

so that as t increases, we converge closer to the optimum Θ_*^y; that is a well-known result [76].

Now, we show how the addition of the loss term on the sensitive attribute, $\mathcal{L}^s$, prevents the convex classifier from fully converging. Let $\beta^s > 0$, then

$$\begin{aligned} dV_t &= -(\Theta_t - \Theta_*^y)^\top \nabla_\Theta \beta^y \mathcal{L}^y(\Theta_t) dt - \beta^s (\Theta_t - \Theta_*^y)^\top \nabla_\Theta \mathcal{L}^s(\Theta_t) dt \\ &\leq -\beta^y \mu^{\mathrm{y}} ||\Theta_t - \Theta_*^y||_2^2 dt + \beta^s \big(\mathcal{L}^s(\Theta_*^y) - \mathcal{L}^s(\Theta_t)\big) dt, \end{aligned}$$

where we use convexity of $\mathcal{L}^y$ and that by convexity[2] of $\mathcal{L}^s$ it holds $(\Theta_*^y - \Theta_t)^\top \nabla_\Theta \mathcal{L}^s(\Theta_t) \leq \mathcal{L}^s(\Theta_*^y) - \mathcal{L}^s(\Theta_t)$. Integrating, we obtain

$$V_t \leq e^{-t\beta^y \mathcal{L}^y(\Theta_t)} V_0 + \beta^s \int_0^t e^{\beta^s \mathcal{L}^y(\Theta_t)(u-t)} \big(\mathcal{L}^s(\Theta_*^y) - \mathcal{L}^s(\Theta_u)\big) du.$$

This result shows that exact convergence is only obtained if $\Theta_*^y = \Theta_*^s$ since in this case, by the convexity of $\mathcal{L}^s$, the right-most term would have been upper-bounded by zero as Θ_*^y would have been the optimizer also for $\mathcal{L}^s$. Otherwise, we only converge to a neighborhood of the optimum Θ_*^y, where the size of the neighborhood is governed by β^s and the properties of $\mathcal{L}^s$.

As a summary, the trade-off that we face for a classifier with *limited* capacity is that: if we also optimize for the sensitive attribute, we will only ever converge to a neighborhood of the optimum for the target attribute. While these results hold for a convex setting and are simplistic in nature, they give us a basic intuition into the idea that when the attributes y and s are somehow *correlated*, the output $\hat{y}$ can better encode both tasks. This result also gives us the intuition that when we have *uncorrelated* attributes, we should need classifiers with more capacity to cover the payoff for not converging to the optimal point of target attribute.

B PSEUDOCODE OF PARAMETERIZED ATTACK

In Algorithm 1 we show the pseudo-code of the training process we presented in Section 4.3.1.

[2] In a convex function, for all x and y we have: $f(x) \geq f(y) + f'(y)(x-y)$.

Algorithm 1 Training an HBC classifier and its corresponding attack based on information bottleneck formulation in Section 4.3.1

Input: $\mathcal{F}$: the model chosen as the classifier, $\mathcal{G}$: the model chosen as the attack , $\mathbb{D}^{train}$: training dataset, $(\beta^x, \beta^y, \beta^s)$: trade-off multipliers, E: number of training epochs, K: batch size.
Output: Updated $\mathcal{F}$ and $\mathcal{G}$.
Random initialization of $\mathcal{F}$ and $\mathcal{G}$.
for $e : 1, \ldots, E$ **do**
  for $b : 1, \ldots, |\mathbb{D}^{train}|/K$ **do**
    $(\mathbf{X}, \mathbf{Y}, \mathbf{S}) \leftarrow$ a random batch of K samples $(\mathbf{x}, y, s) \sim \mathbb{D}^{train}$
    $\hat{\mathbf{Y}} = \mathcal{F}(\mathbf{X})$
    $\hat{\mathbf{S}} = \mathcal{G}(\hat{\mathbf{Y}})$
    $\mathsf{H}(S|\hat{\mathbf{Y}}) = -\sum_{k=1}^{K} \sum_{i=0}^{S-1} \mathbb{I}_{(s^k=i)} log(\hat{s}_i^k)$
    Update $\mathcal{G}$ using SGD via loss function $\mathcal{L}^{\mathcal{G}} = \mathsf{H}(S|\hat{\mathbf{Y}})$
    $\hat{\mathbf{S}} = \mathcal{G}(\hat{\mathbf{Y}})$
    $\mathsf{H}(S|\hat{\mathbf{Y}}) = -\sum_{k=1}^{K} \sum_{i=0}^{S-1} \mathbb{I}_{(s^k=i)} log(\hat{s}_i^k)$
    $\mathsf{H}(Y|\hat{\mathbf{Y}}) = -\sum_{k=1}^{K} \sum_{i=0}^{Y-1} \mathbb{I}_{(y^k=i)} log(\hat{y}_i^k)$
    $\mathsf{H}(\hat{\mathbf{Y}}) = -\sum_{k=1}^{K} \sum_{i=0}^{Y-1} \hat{y}_i^k log(\hat{y}_i^k)$
    Update $\mathcal{F}$ using SGD via loss function $\mathcal{L}^{\mathcal{F}} = \beta^x \mathsf{H}(\hat{\mathbf{y}}) + \beta^y \mathsf{H}(y|\hat{\mathbf{y}}) + \beta^s \mathsf{H}(s|\hat{\mathbf{y}})$
  end for
end for

Table 7: The honesty (δ^y) and curiosity (δ^s) of an HBC classifier trained via regularized attack, where target attribute (y) is Race and sensitive attribute (s) is Age. Values are in percentage (%).

			$S=2$		$S=3$		$S=4$		$S=5$	
			δ^y	δ^s	δ^y	δ^s	δ^y	δ^s	δ^y	δ^s
$Y=2$	NC	(1.,0.)	87.63 ± .19	*62.37 ± .41*	87.63 ± .19	*51.47 ± .28*	87.63 ± .19	*37.80 ± .56*	87.63 ± .19	*32.26 ± .07*
	RawHBC	(.7,.3)	87.70 ± .26	81.42 ± .86	87.82 ± .19	79.58 ± .84	88.17 ± .12	65.67 ± .79	87.94 ± .18	58.15 ± 1.6
	SoftHBC	(.7,.3)	87.22 ± .19	67.22 ± .40	87.57 ± .33	54.03 ± .78	87.26 ± .16	40.86 ± .22	87.24 ± .34	36.60 ± 2.5
		(.5,.5)	86.58 ± .75	71.78 ± .17	82.97 ± 1.6	65.53 ± 1.0	80.67 ± 1.2	48.44 ± .87	83.41 ± .31	53.96 ± 1.3
$Y=3$	NC	(1.,0.)	85.42 ± .46	*62.25 ± .87*	85.42 ± .46	*51.97 ± .43*	85.42 ± .46	*38.09 ± .77*	85.42 ± .46	*32.34 ± .45*
	RawHBC	(.7,.3)	86.11 ± .42	82.14 ± .35	86.13 ± .31	78.09 ± .88	86.33 ± .08	65.56 ± .94	86.11 ± .34	58.95 ± 1.3
	SoftHBC	(.7,.3)	85.25 ± .26	72.47 ± 2.4	85.36 ± .61	64.52 ± 1.0	85.31 ± .38	53.03 ± .90	85.19 ± .10	45.77 ± .51
		(.5,.5)	84.12 ± .27	81.16 ± .23	76.85 ± .38	79.74 ± .64	76.00 ± .31	66.05 ± .46	75.37 ± .27	60.10 ± .49
$Y=4$	NC	(1.,0.)	81.57 ± .33	*64.99 ± .70*	81.57 ± .33	*54.12 ± .58*	81.57 ± .33	*41.13 ± .15*	81.57 ± .33	*35.56 ± .47*
	RawHBC	(.7,.3)	81.69 ± .12	81.33 ± .38	81.08 ± .57	77.57 ± .54	81.26 ± .35	64.44 ± .54	81.15 ± .22	58.93 ± .82
	SoftHBC	(.7,.3)	81.20 ± .21	75.17 ± .35	81.27 ± .48	69.14 ± .07	81.09 ± .10	55.63 ± .11	81.21 ± .08	51.28 ± .52
		(.5,.5)	78.09 ± .12	81.02 ± .58	71.89 ± .27	78.72 ± .39	70.43 ± .61	65.99 ± .75	69.81 ± .45	58.89 ± .37
$Y=5$	NC	(1.,0.)	80.66 ± .24	*66.29 ± .39*	80.66 ± .24	*56.64 ± .25*	80.66 ± .24	*44.42 ± .76*	80.66 ± .24	*38.13 ± .73*
	RawHBC	(.7,.3)	81.05 ± .40	80.93 ± .78	80.78 ± .66	78.48 ± .26	80.00 ± .45	65.16 ± .46	80.87 ± .70	58.71 ± .24
	SoftHBC	(.7,.3)	80.94 ± .35	79.59 ± .27	80.59 ± .08	76.57 ± .28	80.37 ± .09	62.76 ± .47	80.52 ± .06	55.75 ± .52
		(.5,.5)	80.03 ± .22	81.46 ± .23	72.36 ± .55	78.57 ± .17	70.48 ± 1.6	66.32 ± .41	70.88 ± .66	58.02 ± 1.2

C ADDITIONAL EXPERIMENTAL RESULTS

Table 7 and Table 8 show experimental results for settings where target and sensitive attribute are reversed, compared to Table 3 and Table 4 in Section 5. We can observe similar patterns in these results as well, confirming our findings explained in Section 5.

Table 8: Gender and Race. Classification accuracy (%) on UTKFace test set for race vs. gender. In each experiment, there are 4 different tasks based on the size Y and S. Empty cells (—) are due to the incapability of regularized attacks for $S > 2$.

	Setting	Attack	(β^y, β^s)	$S = 2$ δ^y	$S = 2$ δ^s	$S = 3$ δ^y	$S = 3$ δ^s	$S = 4$ δ^y	$S = 4$ δ^s	$S = 5$ δ^y	$S = 5$ δ^s
$Y = 2$	Std	raw [61]	(1., *0.*)	90.23 ± .21	*58.62 ± 1.2*	90.23 ± .21	*47.54 ± 1.1*	90.23 ± .21	*44.21 ± .43*	90.23 ± .21	*43.94 ± .04*
	RawHBC	Parameterized	(.7, .3)	90.18 ± .11	86.91 ± .32	90.56 ± .23	81.66 ± .24	90.24 ± .09	74.04 ± .48	90.28 ± .35	74.01 ± .89
	SoftHBC	Parameterized	(.7, .3)	90.59 ± .07	56.21 ± .00	89.92 ± .11	46.26 ± .50	90.37 ± .19	44.31 ± .21	90.14 ± .34	44.93 ± .94
			(.5, .5)	89.74 ± .43	61.60 ± 4.1	86.95 ± .66	52.85 ± 2.3	87.69 ± .48	50.57 ± .53	87.57 ± .76	50.88 ± .78
			(.3, .7)	79.96 ± .80	81.54 ± .60	72.41 ± .55	77.28 ± .45	72.97 ± 7.3	63.90 ± 2.6	72.97 ± 7.3	63.90 ± 2.6
		Regularized	(.7, .3)	90.18 ± .17	73.20 ± .10	—	—	—	—	—	—
			(.5, .5)	89.71 ± .06	82.28 ± .10	—	—	—	—	—	—
			(.3, .7)	87.86 ± .40	84.35 ± .48	—	—	—	—	—	—

D CURIOSITY AND GENERALIZATION

Underlying the capabilities of the HBC classifiers is the ability of the output $\hat{\mathbf{y}}$ to encode multiple attributes. Such capability also forms the foundation for *representation learning* [5], *multi-task learning* [12] and *disentanglement of factors* [34].

It is well-known that added noise during training in the form of e.g. Dropout, improves the generalization performance of a model [68]. These observations coincide with results obtained in [37], where it is shown that differential privacy is not an effective protection against property inference attacks; even more so, they may improve the performance by keeping the model from overfitting on the main task. Information bottleneck (IB) is used for understanding the nature of intermediate representations produced by the hidden layers of DNNs during training [55, 59, 66].

In Section 4.3.1, we discussed that formulation in Eq. (4) shows a trade-off: compressing the information included in $\mathbf{x}$, while encoding as much information about y and s as possible into $\hat{\mathbf{y}}$. In other words, IB states that the optimal HBC classifier must keep the information in $\mathbf{x}$ that is useful for y and s while compressing other, irrelevant information. On the other hand, a useful classifier is one that does not *overfit* on its training dataset, meaning that the *generalization gap*[3] is small.

Here, we discuss how, under sufficient "similarity" between the target and sensitive attribute, the addition of the sensitive attribute into the classifier's optimization objective can even *improve* the generalization capabilities for the classification of target attribute [30]. Specifically, the addition of another attribute s while training for the target attribute y can reduce the generalization gap as the addition of another attribute can act as a *regularizer*, which places an inductive bias on the learning of the target attribute and guides the model towards learning more general and discerning features [7, 78].

Previous work (*e.g.,* Theorem 1 in [68]) has shown that the generalization gap can be upper-bounded by the amount of compression happening in the classifier, *i.e.,* the mutual information between the internal representations learned by the classifier and the input data. Basically, the compression forces the classifier to throw away information unrelated to the target task (*e.g.,* noises). Moreover, [30] shows that training a classifier for more than one attribute can bring a major benefit in situations where the single-attribute classifier "underperforms" on the target task, *e.g.,* due to the lack of samples in the training data or the presence of noise. Particularly this happens when attributes are related—where the relatedness is measured through the alignment of the input features—and the data for the second attribute is of good quality [30].

Our observation from Section 5 is that, while properties Age, Race, and Gender are not necessarily correlated properties, they are all coarse-grained properties for which understanding of the full image is required (as opposed to fine-grained properties present only in part of the image). As observed from the results, in certain settings, including curiosity in the classifier can improve the performance on the main task. Therefore, because all properties are sufficiently coarse-grained, learning in a multi-task framework can keep the network from

[3] The difference between the classifier's performance on its training data and the classifier's performance on unseen test data is called the generalization gap of classifier.

Table 9: The details of how we assign *age* label to each image in UTKFace.

Label	Number of Classes 2	3	4	5	Samples
0	a ≤ 30	a ≤ 20	a ≤ 21	a ≤ 19	4593
1	30 > a	20 < a ≤ 35	21 < a ≤ 29	19 < a ≤ 26	5241
2		35 < a	29 < a ≤ 45	26 < a ≤ 34	4393
3			45 < a	34 < a ≤ 49	4491
4				49 < a	4987

overfitting on details by enforcing it to learn certain general, coarse-grained features and thus perform better in the main task.

Putting all of this together, it can be expected that by training an HBC classifier, we can not only reduce the generalization gap for the target attribute, but also compress the learned representations, which means decrease (or at least not increase) the entropy of the representations. Thus, in situations where the curiosity of the classifier *improves* generalization, the recognition of an HBC classifier is even more challenging.

E MORE MOTIVATIONAL EXAMPLES

As some other motivational examples, we can consider:

(1) A sentiment analysis application that can be run on the user's browser and, taking a text, it can help the user to estimate how their text may sound to the readers: *e.g.,* $\hat{\mathbf{y}} = [p_{positive}, p_{neutral}, p_{negative}]$. The server may ask for collecting only outputs, *e.g.,* to improve their service, but server might try to infer if the text includes specific sensitive words or if the text falls into a certain category; by making an HBC sentiment classifier.

(2) An activity recognition app, from an insurance company, can be run on the user's smartwatch and analyzing motion data. This app can provide an output $\hat{\mathbf{y}} = [p_{walk}, p_{run}, p_{sit}, p_{sleep}]$. The server may ask for only collecting $\hat{\mathbf{y}}$ to offer the user further health statistics or advice, but it may also try to infer if the user's body mass index (BMI) is more than or below a threshold.

(3) Automatic speech recognition (ASR) is the core technology for all voice assistants, such as Amazon Alexa, Apple Siri, Google Assistant, Microsoft Cortana, to provide specialized services to their users, *e.g.,* playing a song, reporting weather condition, or ordering food. ASR models, at the top, have a softmax layer that estimates a probability distribution on a vocabulary with a particular length, *e.g.,* 26 letters in the English alphabet plus some punctuation symbols. Service providers usually perform post-processing on the output of the ASR model, that is the probable letter for each time frame, to combine them and predict the most probable word and further the actual sentence uttered by the user. Despite the fact that they can be asked to run their ASR model at the user's side, but they can argue that the post-processing ASR's outputs is crucial for providing a good service, and these outputs are supposed to only predict what the user is asking for (*i.e.,* the uttered sentence) and not any other information, *e.g.,* the age, gender, or race of the user. However, an HBC ASR might secretly violate its user's privacy.

F DATASETS

F.1 Details of CelebA Dataset

Figure 10 shows some sample images from CelebA dataset [32]. While CelebA does not have any attributes with the number of classes more than 2, we utilize three mutually-exclusive binary attributes BlackHair, BlondHair, BrownHair to build a 3-class attribute of HairColor. This still gives us a dataset of more than 115K images that fall into one of these categories.

Table 10: The details of how we assign *race* label to each image in UTKFace. W:White, B:Black, A:Asian, I:Indian, or O:others.

	Number of Classes				
Label	2	3	4	5	Samples
0	W	W	W	W	10078
1	BAIO	A	B	B	4526
2		BIO	I	A	3434
3			AO	I	3975
4				O	1692

Figure 10: Sample images from CelebA [32]. Columns (from left to right) show Male, Smiling, Attractive, High Cheekbones, and Heavy Makeup attributes. The first and second rows show Black Hair with and without the other corresponding column's attribute, respectively. Similarly, the third and fourth rows Blond Hair, and fifth and sixth rows Brown Hair attributes.

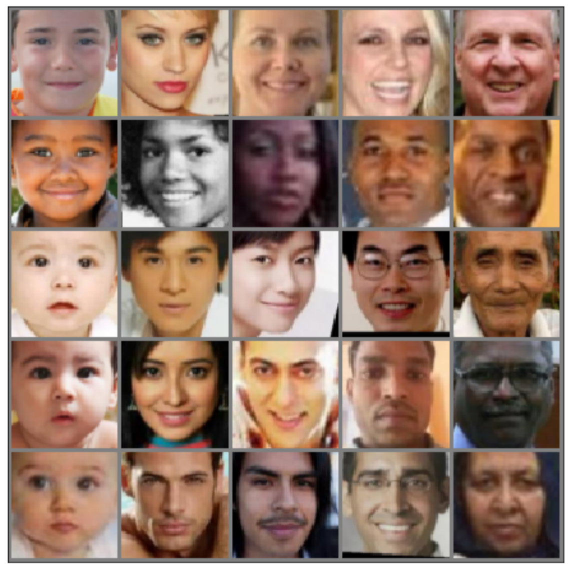

Figure 11: Sample images from UTKFace [79]. Each row shows images labeled with the same race (White, Black, Asian, Indian, and others), and each column shows one of the five age groups ([0-19,20-26,27-34,35-49, 50-100].

F.2 Details of UTKFace Dataset

Figure 11 shows some sample images from UTKFace dataset [79]. Each image, collected from the Internet, has three attributes: (1) gender (male or female), (2) race (White, Black, Asian, Indian, or others), and (3) age (from 0 to 116). Gender is a binary label and there are 12391 vs. 11314 images with male vs. female labels. Table 9 and Table 10 show how we assign age and race labels to each image in UTKFace for different number of classes used in our evaluations. For age label we choose categories such that classes become balanced, in terms of the number

Table 11: The implemented model as classifier $\mathcal{F}$.

Layer Type	Output Size	Number of Parameters
Conv2d(kernel=2, stride=2)	128, 32, 32	1,664
BatchNorm2d	128, 32, 32	256
LeakyReLU(slope=0.01)	128, 32, 32	0
Dropout(p=0.2)	128, 32, 32	0
Conv2d(kernel=2, stride=2)	128, 16, 16	65,664
BatchNorm2d	128, 16, 16	256
LeakyReLU(slope=0.01)	128, 16, 16	0
Dropout(p=0.2)	128, 16, 16	0
Conv2d(kernel=2, stride=2)	64, 8, 8	32,832
BatchNorm2d	64, 8, 8	128
LeakyReLU(slope=0.01)	64, 8, 8	0
Dropout(p=0.2)	64, 8, 8	0
Conv2d(kernel=2, stride=2)	64, 4, 4	16,448
BatchNorm2d	64, 4, 4	128
LeakyReLU(slope=0.01)	64, 4, 4	0
Dropout(p=0.2)	64, 4, 4	0
Linear	128	131,200
LeakyReLU(slope=0.01)	128	0
Dropout(p=0.5)	128	0
Linear	Y	128×Y+Y

Table 12: The implemented model as attack $\mathcal{G}$.

Layer Type	Output Size	Number of Parameters
Linear	20×Y	80×Y
LeakyReLU(slope=0.01)	20×Y	0
Dropout(p=0.25)	20×Y	0
Linear	10×Y	610×Y
LeakyReLU(slope=0.01)	10×Y	0
Dropout(p=0.25)	10×Y	0
Linear	S	10×Y×S+S

of samples. For race label, we are not that free (like age), thus we choose categories such that classes do not become highly unbalanced and we try to keep samples in each class as similar to each other as possible.

G MODEL ARCHITECTURES

For all experiments reported in this paper, we use PyTorch [48] that is an open-source library for the implementation of deep neural networks [29, 56]. Table 11 and Table 12 show the details of neural network architecture that are implemented as classifier $\mathcal{F}$ and attack $\mathcal{G}$ (see Figure 6).