Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures

Matt Fredrikson
Carnegie Mellon University

Somesh Jha
University of Wisconsin–Madison

Thomas Ristenpart
Cornell Tech

ABSTRACT

Machine-learning (ML) algorithms are increasingly utilized in privacy-sensitive applications such as predicting lifestyle choices, making medical diagnoses, and facial recognition. In a model inversion attack, recently introduced in a case study of linear classifiers in personalized medicine by Fredrikson et al. [13], adversarial access to an ML model is abused to learn sensitive genomic information about individuals. Whether model inversion attacks apply to settings outside theirs, however, is unknown.

We develop a new class of model inversion attack that exploits confidence values revealed along with predictions. Our new attacks are applicable in a variety of settings, and we explore two in depth: decision trees for lifestyle surveys as used on machine-learning-as-a-service systems and neural networks for facial recognition. In both cases confidence values are revealed to those with the ability to make prediction queries to models. We experimentally show attacks that are able to estimate whether a respondent in a lifestyle survey admitted to cheating on their significant other and, in the other context, show how to recover recognizable images of people's faces given only their name and access to the ML model. We also initiate experimental exploration of natural countermeasures, investigating a privacy-aware decision tree training algorithm that is a simple variant of CART learning, as well as revealing only rounded confidence values. The lesson that emerges is that one can avoid these kinds of MI attacks with negligible degradation to utility.

1. INTRODUCTION

Computing systems increasingly incorporate machine learning (ML) algorithms in order to provide predictions of lifestyle choices [6], medical diagnoses [20], facial recognition [1], and more. The need for easy "push-button" ML has even prompted a number of companies to build ML-as-a-service cloud systems, wherein customers can upload data sets, train classifiers or regression models, and then obtain access to perform prediction queries using the trained model — all over easy-to-use public HTTP interfaces. The features used by these models, and queried via APIs to make predictions, often represent sensitive information. In facial recognition, the features are the individual pixels of a picture of a person's face. In lifestyle surveys, features may contain sensitive information, such as the sexual habits of respondents.

In the context of these services, a clear threat is that providers might be poor stewards of sensitive data, allowing training data or query logs to fall prey to insider attacks or exposure via system compromises. A number of works have focused on attacks that result from access to (even anonymized) data [18,29,32,38]. A perhaps more subtle concern is that the ability to make prediction queries might enable adversarial *clients* to back out sensitive data. Recent work by Fredrikson et al. [13] in the context of genomic privacy shows a *model inversion attack* that is able to use black-box access to prediction models in order to estimate aspects of someone's genotype. Their attack works for any setting in which the sensitive feature being inferred is drawn from a small set. They only evaluated it in a single setting, and it remains unclear if inversion attacks pose a broader risk.

In this paper we investigate commercial ML-as-a-service APIs. We start by showing that the Fredrikson et al. attack, even when it is computationally tractable to mount, is not particularly effective in our new settings. We therefore introduce new attacks that infer sensitive features used as inputs to decision tree models, as well as attacks that recover images from API access to facial recognition services. The key enabling insight across both situations is that we can build attack algorithms that exploit confidence values exposed by the APIs. One example from our facial recognition attacks is depicted in Figure 1: an attacker can produce a recognizable image of a person, given only API access to a facial recognition system and the name of the person whose face is recognized by it.

ML APIs and model inversion. We provide an overview of contemporary ML services in Section 2, but for the purposes of discussion here we roughly classify client-side access as being either *black-box* or *white-box*. In a black-box setting, an adversarial client can make prediction queries against a model, but not actually download the model description. In a white-box setting, clients are allowed to download a description of the model. The new generation of ML-as-a-service systems—including general-purpose ones such as BigML [4] and Microsoft Azure Learning [31]—allow data owners to specify whether APIs should allow white-box or black-box access to their models.

CCS'15, October 12–16, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813677 .

Figure 1: An image recovered using a new model inversion attack (left) and a training set image of the victim (right). The attacker is given only the person's name and access to a facial recognition system that returns a class confidence score.

Consider a model defining a function f that takes input a feature vector $\mathbf{x}_1, \ldots, \mathbf{x}_d$ for some feature dimension d and outputs a prediction $y = f(\mathbf{x}_1, \ldots, \mathbf{x}_d)$. In the model inversion attack of Fredrikson *et al.* [13], an adversarial client uses black-box access to f to infer a sensitive feature, say $\mathbf{x}_1$, given some knowledge about the other features and the dependent value y, error statistics regarding the model, and marginal priors for individual variables. Their algorithm is a maximum a posteriori (MAP) estimator that picks the value for $\mathbf{x}_1$ which maximizes the probability of having observed the known values (under some seemingly reasonable independence assumptions). To do so, however, requires computing $f(\mathbf{x}_1, \ldots, \mathbf{x}_d)$ for every possible value of $\mathbf{x}_1$ (and any other unknown features). This limits its applicability to settings where $\mathbf{x}_1$ takes on only a limited set of possible values.

Our first contribution is evaluating their MAP estimator in a new context. We perform a case study showing that it provides only limited effectiveness in estimating sensitive features (marital infidelity and pornographic viewing habits) in decision-tree models currently hosted on BigML's model gallery [4]. In particular the false positive rate is too high: our experiments show that the Fredrikson et al. algorithm would incorrectly conclude, for example, that a person (known to be in the training set) watched pornographic videos in the past year almost 60% of the time. This might suggest that inversion is not a significant risk, but in fact we show new attacks that can significantly improve inversion efficacy.

White-box decision tree attacks. Investigating the actual data available via the BigML service APIs, one sees that model descriptions include more information than leveraged in the black-box attack. In particular, they provide the count of instances from the training set that match each path in the decision tree. Dividing by the total number of instances gives a confidence in the classification. While a priori this additional information may seem innocuous, we show that it can in fact be exploited.

We give a new MAP estimator that uses the confidence information in the white-box setting to infer sensitive information with *no false positives* when tested against two different BigML decision tree models. This high precision holds for target subjects who are known to be in the training data, while the estimator's precision is significantly worse for those not in the training data set. This demonstrates that publishing these models poses a privacy risk for those contributing to the training data.

Our new estimator, as well as the Fredrikson et al. one, query or run predictions a number of times that is linear in the number of possible values of the target sensitive feature(s). Thus they do not extend to settings where features have exponentially large domains, or when we want to invert a large number of features from small domains.

Extracting faces from neural networks. An example of a tricky setting with large-dimension, large-domain data is facial recognition: features are vectors of floating-point pixel data. In theory, a solution to this large-domain inversion problem might enable, for example, an attacker to use a facial recognition API to recover an image of a person given just their name (the class label). Of course this would seem impossible in the black-box setting if the API returns answers to queries that are just a class label. Inspecting facial recognition APIs, it turns out that it is common to give floating-point confidence measures along with the class label (person's name). This enables us to craft attacks that cast the inversion task as an optimization problem: *find the input that maximizes the returned confidence, subject to the classification also matching the target.* We give an algorithm for solving this problem that uses gradient descent along with modifications specific to this domain. It is efficient, despite the exponentially large search space: reconstruction completes in as few as 1.4 seconds in many cases, and in 10–20 minutes for more complex models in the white-box setting.

We apply this attack to a number of typical neural network-style facial recognition algorithms, including a softmax classifier, a multilayer perceptron, and a stacked denoising autoencoder. As can be seen in Figure 1, the recovered image is not perfect. To quantify efficacy, we perform experiments using Amazon's Mechanical Turk to see if humans can use the recovered image to correctly pick the target person out of a line up. Skilled humans (defined in Section 5) can correctly do so for the softmax classifier with close to 95% accuracy (average performance across all workers is above 80%). The results are worse for the other two algorithms, but still beat random guessing by a large amount. We also investigate related attacks in the facial recognition setting, such as using model inversion to help identify a person given a blurred-out picture of their face.

Countermeasures. We provide a preliminary exploration of countermeasures. We show empirically that simple mechanisms including taking sensitive features into account while using training decision trees and rounding reported confidence values can drastically reduce the effectiveness of our attacks. We have not yet evaluated whether MI attacks might be adapted to these countermeasures, and this suggests the need for future research on MI-resistant ML.

Summary. We explore privacy issues in ML APIs, showing that confidence information can be exploited by adversarial clients in order to mount model inversion attacks. We provide new model inversion algorithms that can be used to infer sensitive features from decision trees hosted on ML services, or to extract images of training subjects from facial recognition models. We evaluate these attacks on real data, and show that models trained over datasets involving survey respondents pose significant risks to feature confidentiality, and that recognizable images of people's faces can be extracted from facial recognition models. We evaluate preliminary countermeasures that mitigate the attacks we develop, and might help prevent future attacks.

2. BACKGROUND

Our focus is on systems that incorporate machine learning models and the potential confidentiality threats that arise when adversaries can obtain access to these models.

ML basics. For our purposes, an ML model is simply a deterministic function $f\colon \mathbb{R}^d \mapsto Y$ from d *features* to a set of *responses* Y. When Y is a finite set, such as the names of people in the case of facial recognition, we refer to f as a *classifier* and call the elements in Y the *classes*. If instead $Y = \mathbb{R}$, then f is a *regression model* or simply a *regression*.

Many classifiers, and particularly the facial recognition ones we target in Section 5, first compute one or more regressions on the feature vector for each class, with the corresponding outputs representing estimates of the likelihood that the feature vector should be associated with a class. These outputs are often called *confidences*, and the classification is obtained by choosing the class label for the regression with highest confidence. More formally, we define f in these cases as the composition of two functions. The first is a function $\tilde{f}\colon \mathbb{R}^d \mapsto [0,1]^m$ where m is a parameter specifying the number of confidences. For our purposes $m = |Y| - 1$, i.e., one less than the number of class labels. The second function is a selection function $t\colon [0,1]^m \to Y$ that, for example when $m = 1$, might output one label if the input is above 0.5 and another label otherwise. When $m > 1$, t may output the label whose associated confidence is greatest. Ultimately $f(\mathbf{x}) = t(\tilde{f}(\mathbf{x}))$. It is common among APIs for such models that classification queries return both $f(\mathbf{x})$ as well as $\tilde{f}(\mathbf{x})$. This provides feedback on the model's confidence in its classification. Unless otherwise noted we will assume both are returned from a query to f, with the implied abuse of notation.

One generates a model f via some (randomized) training algorithm train. It takes as input labeled training data db, a sequence of $(d+1)$-dimensional vectors $(\mathbf{x}, y) \in \mathbb{R}^d \times Y$ where $\mathbf{x} = \mathbf{x}_1, \ldots, \mathbf{x}_d$ is the set of features and y is the label. We refer to such a pair as an instance, and typically we assume that instances are drawn independently from some prior distribution that is joint over the features and responses. The output of train is the model[1] f and some auxiliary information *aux*. Examples of auxiliary information might include error statistics and/or marginal priors for the training data.

ML APIs. Systems that incorporate models f will do so via well-defined application-programming interfaces (APIs). The recent trend towards ML-as-a-service systems exemplifies this model, where users upload their training data db and subsequently use such an API to query a model trained by the service. The API is typically provided over HTTP(S). There are currently a number of such services, including Microsoft Machine Learning [31], Google's Prediction API [16], BigML [4], Wise.io [40], that focus on analytics. Others [21,26,36] focus on ML for facial detection and recognition.

Some of these services have marketplaces within which users can make models or data sets available to other users. A model can be white-box, meaning anyone can download a description of f suitable to run it locally, or black-box, meaning one cannot download the model but can only make prediction queries against it. Most services charge by the number of prediction queries made [4,16,31,36].

Threat model. We focus on settings in which an adversarial client seeks to abuse access to an ML model API. The adversary is assumed to have whatever information the API exposes. In a *white-box* setting this means access to download a model f. In a *black-box* setting, the attacker has only the ability to make prediction queries on feature vectors of the adversary's choosing. These queries can be adaptive, however, meaning queried features can be a function of previously retrieved predictions. In both settings, the adversary obtains the auxiliary information *aux* output by training, which reflects the fact that in ML APIs this data is currently most often made public.[2]

We will focus on contexts in which the adversary does *not* have access to the training data db, nor the ability to sample from the joint prior. While in some cases training data is public, our focus will be on considering settings with confidentiality risks, where db may be medical data, lifestyle surveys, or images for facial recognition. Thus, the adversary's only insight on this data is indirect, through the ML API.

We do not consider malicious service providers, and we do not consider adversarial clients that can compromise the service, e.g., bypassing authentication somehow or otherwise exploiting a bug in the server's software. Such threats are important for the security of ML services, but already known as important issues. On the other hand, we believe that the kinds of attacks we will show against ML APIs are more subtle and unrecognized as threats.

3. THE FREDRIKSON ET AL. ATTACK

We start by recalling the generic model inversion attack for target features with small domains from Fredrikson et al. [13].

Fredrikson et al. considered a linear regression model f that predicted a real-valued suggested initial dose of the drug Warfarin using a feature vector consisting of patient demographic information, medical history, and genetic markers.[3] The sensitive attribute was considered to be the genetic marker, which we assume for simplicity to be the first feature $\mathbf{x}_1$. They explored model inversion where an attacker, given white-box access to f and auxiliary information $\mathsf{side}(\mathbf{x}, y) \stackrel{\text{def}}{=} (\mathbf{x}_2, \ldots, \mathbf{x}_t, y)$ for a patient instance $(\mathbf{x}, y)$, attempts to infer the patient's genetic marker $\mathbf{x}_1$.

Figure 2 gives their inversion algorithm. Here we assume *aux* gives the empirically computed standard deviation σ for a Gaussian error model err and marginal priors $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_t)$. The marginal prior $\mathbf{p}_i$ is computed by first partitioning the real line into disjoint buckets (ranges of values), and then letting $\mathbf{p}_i(v)$ for each bucket v be the the number of times $\mathbf{x}_i$ falls in v over all $\mathbf{x}$ in db divided by the number of training vectors $|\mathsf{db}|$.

In words, the algorithm simply completes the target feature vector with each of the possible values for $\mathbf{x}_1$, and then computes a weighted probability estimate that this is the correct value. The Gaussian error model will penalize val-

[1] We abuse notation and write f to denote both the function and an efficient representation of it.

[2] For example, BigML.io includes this data on web pages describing black-box models in their model gallery [4].

[3] The inputs were processed in a standard way to make nominal valued data real-valued. See [13] for details.

adversary $\mathcal{A}^f(\mathsf{err}, \mathbf{p}_i, \mathbf{x}_2, \ldots, \mathbf{x}_t, y)$:

1: **for** each possible value v of $\mathbf{x}_1$ **do**
2: $\quad \mathbf{x}' = (v, \mathbf{x}_2, \ldots, \mathbf{x}_t)$
3: $\quad \mathbf{r}_v \leftarrow \mathsf{err}(y, f(\mathbf{x}')) \cdot \prod_i \mathbf{p}_i(\mathbf{x}_i)$
4: Return $\arg\max_v \mathbf{r}_v$

Figure 2: Generic inversion attack for nominal target features.

ues of $\mathbf{x}_1$ that force the prediction to be far from the given label y.

As argued in their work, this algorithm produces the least-biased maximum a posteriori (MAP) estimate for $\mathbf{x}_1$ given the available information. Thus, it minimizes the adversary's misprediction rate. They only analyzed its efficacy in the case of Warfarin dosing, showing that the MI algorithm above achieves accuracy in predicting a genetic marker close to that of a linear model trained directly from the original data set.

It is easy to see that their algorithm is, in fact, black-box, and in fact agnostic to the way f works. That means it is potentially applicable in other settings, where f is not a linear regression model but some other algorithm. An obvious extension handles a larger set of unknown features, simply by changing the main loop to iterate over all possible combinations of values for the unknown features. Of course this only makes sense for combinations of nominal-valued features, and not truly real-valued features.

However, the algorithm proposed by Fredrikson *et al.* has various limitations. Most obviously, it cannot be used when the unknown features cover an intractably large set. One example of such a setting is facial recognition, where the feature space is huge: in the facial recognition examples considered later $d \approx 10,304$ with each feature a real number in $[0, 1]$. Even if one only wanted to infer a portion of the features this is computationally infeasible. A more subtle issue, explored in the next section, is that even when efficient it may not be effective.

It turns out that we can give attacks that overcome both issues above by giving inversion attacks that utilize the *confidence information* revealed by ML APIs. To do so will require new techniques, which we explore using as case studies decision trees (the next section) and facial recognition (Section 5).

4. MAP INVERTERS FOR TREES

We turn now to inversion attacks on decision trees. This type of model is used on a wide range of data, and is often favored because tree-structured rules are easy for users to understand. There are two types of decision trees common in the literature: *classification* (where the class variable is discrete), and *regression* (where the class variable is continuous). In the following, we focus on classification trees.

Decision tree background. A decision tree model recursively partitions the feature space into disjoint *regions* $R_1, \ldots, R_m$. Predictions are made for an instance $(\mathbf{x}, y)$ by finding the region containing $\mathbf{x}$, and returning the most likely value for y observed in the training data within that region.

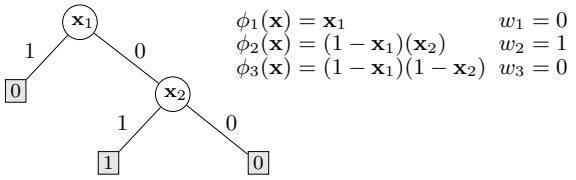


Figure 3: Decision tree for the formula $y = \neg\mathbf{x}_1 \wedge \mathbf{x}_2$.

We characterize trees mathematically as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}), \text{ where } \phi_i(\mathbf{x}) \in \{0, 1\}$$

where each *basis function* ϕ_i is an indicator for R_i, and w_i corresponds to the most common response observed in the training set within R_i. A simple example is shown in Figure 3. Notice that there is exactly one basis function for each path through the tree, and that a given $\mathbf{x}$ will "activate" only one ϕ_i (and thus traverse only one path through the tree) because the basis functions partition the feature space.

Decision trees can be extended to return confidence measures for classifications by changing the form the w_i coefficients take. This is usually accomplished by setting w_i to a vector corresponding to the distribution of class labels in a given region, as observed in the training set. Looking at the example in Figure 3, we would set $w_1 = [89, 11]$ if there were 89 training examples with $\mathbf{x}_1 = 1$ and $\mathbf{x}_2 = 0$, and 11 with $\mathbf{x}_1 = 1, \mathbf{x}_2 = 1$. The classification and corresponding confidences are given by:

$$\begin{aligned} f(\mathbf{x}) &= \arg\max_j \left(\textstyle\sum_{i=1}^{m} w_i[j]\phi_i(\mathbf{x})\right), \text{ and} \\ \tilde{f}(\mathbf{x}) &= \left[\frac{w_{i^*}[1]}{\sum_i w_1[i]}, \ldots, \frac{w_{i^*}[|Y|]}{\sum_i w_m[i]}\right] \end{aligned}$$

where i^* in the second formula takes the value in $\{1, \ldots, m\}$ such that $\phi_{i^*}(\mathbf{x}) = 1$.

Decision tree APIs. Several web APIs expose training and querying routines for decision trees to end-users, including Microsoft Machine Learning [31], Wise.io [40], and BigML [4]. Users can upload their datasets to these services, train a decision tree to predict selected variables, and then make the resulting tree available for others to use for free or for charge per-query. We use as running example the API exposed by BigML, as it currently has the largest marketplace of trained models and focuses on decision trees. The results carry over to services with a similar API as well.

BigML allows users to publish trees in either *black-box* or *white-box* mode. In black-box mode, other users are only allowed to query the decision tree for predictions using a REST API. In white-box mode, users are allowed to see the internal structure of the tree, as well as download a JSON representation of it for local use. The amount of available information about the training set therefore varies between the two modes. In both settings, the adversary has access to marginal priors for each feature of the training set (see Section 3 for a description of this), in addition to a *confusion matrix* $\mathbf{C}$ for which $\mathbf{C}_{i,j}$ gives the number of training instances with $y = i$ for which the model predicted label j. In the white-box setting, the attacker also has access to a count n_i of the number of training set instances that match path

ϕ_i in the tree. This allows one to compute the confidence of a particular prediction.

The inversion problem. Fix a tree $f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$ and let $(\mathbf{x}, y)$ be a target instance that will either be from the training data db or not (we will be clear in the following about which setting we are in). We assume for simplicity that there is one sensitive feature, the first, making the target feature set in this case $T = \{1\}$; extending our attacks to more than one feature is straightforward. The side information $\mathsf{side}_\ell(\mathbf{x}, y) = (\mathbf{x}_\ell, \ldots, \mathbf{x}_d, y)$ for some to-be-specified $\ell \geq 2$. We let $K = \{\ell, \ldots, d\}$ be the set of known feature indices, and we abuse notation slightly to let $\mathbf{x}_K$ represent the $d-1$ dimensional vector equal to $\mathbf{x}_\ell, \ldots, \mathbf{x}_d$.

Black-box MI. For black-box MI we turn to the generic algorithm from Section 3 and adapt it to this setting. The main difference from the prior work's setting is that the decision tree models we consider produce discrete outputs, and the error model information is different, being a confusion matrix as opposed to the standard deviation of a Gaussian. For our purposes here, then, we use the confusion matrix $\mathbf{C}$ and define $\mathsf{err}(y, y') \propto \Pr[\, f(\mathbf{x}) = y' \mid y \text{ is the true label} \,]$. In Section 4.1 we evaluate the algorithm of Figure 2, with this error model, showing that it has unsatisfying performance in terms of a prohibitively high false positive rate.

White-box MI. Recall from before that in the white-box setting, we not only assume that the attacker knows each ϕ_i, but also the number of training samples n_i that correspond to ϕ_i. From this, he also knows $N = \sum_{i=1}^{m} n_i$, the total number of samples in the training set.

The known values $\mathbf{x}_K$ induce a set of paths $S = \{s_i\}_{1 \leq i \leq m}$: $S = \{(\phi_i, n_i) \mid \exists \mathbf{x}' \in \mathbb{R}^d \,.\, \mathbf{x}'_K = \mathbf{x}_K \wedge \phi_i(\mathbf{x}')\}$. Each path corresponds to a basis function ϕ_i and a sample count n_i. We let p_i denote n_i/N, and note that each p_i gives us some information about the joint distribution on features used to build the training set. In the following, we write s_i as shorthand for the event that a row drawn from the joint prior traverses the path s_i, i.e., $\Pr[\, s_i \,]$ corresponds to the probability of drawing a row from the joint prior that traverses s_i. Observe that p_i is an empirical estimate for this quantity, derived from the draws used to produce the training set.

Recall that the basis functions partition the feature space, so we know that $\mathbf{x}$ traverses exactly one of the paths in S. Below we denote a specific value for the first feature by v and a specific value for the other $d-1$ features as v_K. We will abuse notation slightly and write $\phi_i(v)$ as shorthand for the indicator function $\mathbb{I}(\exists \mathbf{x}' \in \mathbb{R}^d \,.\, \mathbf{x}'_1 = v \wedge \phi_i(\mathbf{x}'))$. The following estimator characterizes the probability that $\mathbf{x}_1 = v$ given that $\mathbf{x}$ traverses one of the paths $s_1, \ldots, s_m$ and $\mathbf{x}_K = \mathbf{v}_K$:

$$\begin{aligned}
&\Pr[\, \mathbf{x}_1 = v \mid (s_1 \vee \cdots \vee s_m) \wedge \mathbf{x}_K = \mathbf{v}_K \,] \\
&\quad \propto \sum_{i=1}^{m} \frac{p_i \phi_i(v) \cdot \Pr[\, \mathbf{x}_K = \mathbf{v}_K \,] \cdot \Pr[\, \mathbf{x}_1 = v \,]}{\sum_{j=1}^{m} p_j \phi_j(v)} \\
&\quad \propto \frac{1}{\sum_{j=1}^{m} p_j \phi_j(v)} \sum_{1 \leq i \leq m} p_i \phi_i(v) \cdot \Pr[\, \mathbf{x}_1 = v \,] \qquad (1)
\end{aligned}$$

We refer to this as the *white-box with counts* (WBWC) estimator. The adversary then outputs a value for v that maximizes (1) as a guess for $\mathbf{x}_1$. Like the Fredrikson et al. estimator, it returns the MAP prediction given the additional count information.

In the preceding analysis, we assumed that the attacker knew all of $\mathbf{x}$ except $\mathbf{x}_1$. The WBWC estimator can be extended to the general case where the attacker does not know $\mathbf{x}_2, \ldots, \mathbf{x}_l$ (so $\mathbf{x}_K = \{l+1, \ldots, d-1\}$) by summing (1) over the unknown variables.

4.1 Experiments

We applied the black-box and white-box WBWC attacks to decision trees trained on two datasets: FiveThirtyEight's "How Americans Like Their Steak" survey [17], and a subset of the General Social Survey (GSS) focusing on responses related to marital happiness [33]. Each dataset contains rows that correspond to individuals, with attributes corresponding to survey responses. These datasets contain at least one sensitive feature, and have been used to derive models that are available on BigML's marketplace. Additionally, the source data is public, which makes them appropriate surrogates for our study—without source data, we cannot evaluate the effectiveness of our attacks.

FiveThirtyEight survey. In May 2014, Walt Hickey wrote an article for FiveThirtyEight's *DataLab* section that attempted a statistical analysis of the connection between peoples' steak preparation preferences and their propensity for risk-taking behaviors [17]. To support the analysis, FiveThirtyEight commissioned a survey of 553 individuals from SurveyMonkey, which collected responses to questions such as: "Do you ever smoke cigarettes?", "Have you ever cheated on your significant other?", and of course, "How do you like your steak prepared?". Demographic characteristics such as age, gender, household income, education, and census region were also collected. We discarded rows that did not contain responses for the infidelity question or the steak preparation question, resulting in a total of 332 rows for the inversion experiments. We used model inversion on the decision tree learned from this dataset to infer whether each participant responded "Yes" to the question about infidelity.

GSS marital happiness survey. The General Social Survey (GSS) collects detailed information on the demographics, interests, and attitudes of United States residents [37]. We use a subset of the GSS data [33] created by Joseph Price for the purposes of studying various societal effects of pornography. This subset corresponds to 51,020 individuals and 11 variables, including basic demographic information and responses to questions such as, "How happy are you in your marriage?" and "Have you watched X-rated movies in the last year?" We discarded rows that did not contain responses to either of these questions, resulting in 16,127 total rows for the inversion experiments. We use model inversion to infer each participant's response to the latter question.

Summary of results. For both datasets, we were able to identify positive instances of the sensitive variable (i.e., a "Yes" answer to "Have you ever cheated on your significant other?" or "Have you watched X-rated movies in the last year?") with high precision. Key findings are:

- Given white-box access to the BigML trees published by others for these datasets, we are able to predict positive instances with perfect precision, i.e., no false positives.
- Individuals whose responses are used in the training data are at significantly higher risk than individuals not included in the training data. The results are stark: on the FiveThirtyEight survey, white-box inversion yields

on average 593× improved precision and 371× improved recall. Similar results hold for the GSS survey.

- White-box access to decision trees enhances the adversary's advantage. On the BigML tree trained using GSS data, the white-box adversary has a significant boost in precision over the black-box adversary (158% greater) for a modest decrease in recall (32% less). In other words, the white-box adversary is able to identify slightly fewer "Yes" responses than the black-box adversary, but with much better (i.e., perfect) precision.

Our results indicate that releasing a decision tree trained over either of these datasets substantially increases the risk of confidential data leakage for the individuals who provided responses. Details are given below.

Methodology. We ran the white-box and black-box inversion attacks for each row of the datasets described above. To evaluate the risk posed by existing, publicly-available models, we used trees obtained from the BigML service [4]. Because the sensitive feature in each dataset is binary-valued, we use *precision* and *recall* to measure the effectiveness of the attack. In our setting, precision measures the fraction of predicted "Yes" responses that are correct, and recall measures the fraction of "Yes" responses in the dataset that are predicted by the adversary. We also measure the *accuracy* which is defined as the fraction of correct responses. The trees published on BigML used the entire dataset for training. To evaluate the effect of training set inclusion, we trained trees locally by constructing 100 trees using default parameters on randomly-sampled stratified training sets comprised of 50% of the available data. We downloaded the models and ran the experiments locally to avoid placing a burden on BigML's servers. We used a machine with 8 Xeon cores running at 2.5 Ghz, with 16G of memory.

To establish a baseline for comparison, we use three prediction strategies corresponding to the capabilities of a *random* adversary, a *baseline* adversary, and an *ideal* adversary.

- The random adversary has access to no information aside from the domain of the sensitive attribute. This corresponds to an attacker who cannot access a model, and knows nothing about the prior. For both of our datasets, the sensitive feature is binary, so the adversary's best strategy is to flip a fair coin.

- The baseline adversary has access only to the marginal probability distribution for the sensitive attribute, and not the tree or any other information about the training set. The baseline adversary's best strategy is to always guess the most likely value according to the marginal distribution (i.e., its mode, "No" on our data).

- The ideal adversary has access to a decision tree trained from the original dataset to predict the sensitive attribute, and given the known features for an individual, uses the tree to make a prediction.

This strategy for the ideal adversary is the appropriate one in our setting as it inherits the limitations of the model class: because the attack inverts a decision tree to make predictions, we do not expect those predictions to be *more* accurate than ones made by a decision tree trained to predict the sensitive feature in the forward direction, from the same data. This kind of same-model-class comparison was also used in [13].

	FiveThirtyEight			GSS		
algorithm	**acc.**	**prec.**	**rec.**	**acc.**	**prec.**	**rec.**
whitebox	86.4	100.0	21.1	80.3	100.0	0.7
blackbox	85.8	85.7	21.1	80.0	38.8	1.0
random	50.0	50.0	50.0	50.0	50.0	50.0
baseline	82.9	0.0	0.0	82.0	0.0	0.0
ideal	99.8	100.0	98.6	80.3	61.5	2.3

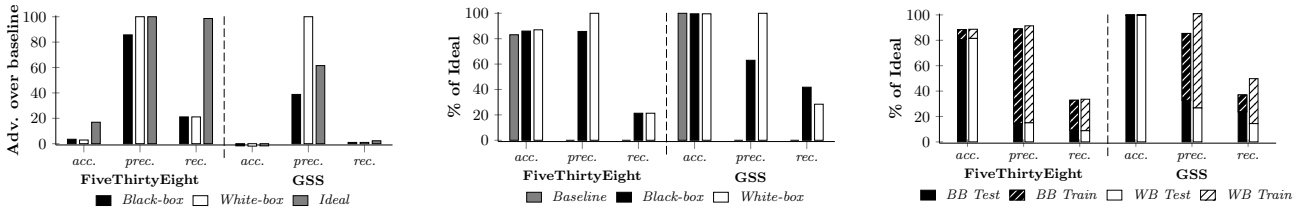
Figure 4: MI results for for BigML models. All numbers shown are percentages.

Performance. The amount of time needed to run both the black-box and white-box attacks is negligible, with the white-box attack being the most expensive. This attack took about 6 milliseconds on average for both datasets, with the most expensive component being path enumeration through a fairly large (about 200 paths) tree. The number of calls needed to run the black-box attack is small: 4 for the FiveThirtyEight dataset (there are 2 unknown binary features), and 2 for the GSS dataset. We conclude that for datasets similar to these, having few unknown features from small domains, the black-box attack is feasible to execute remotely.

Discussion. Figure 4 shows the full results for our experiments. The largest difference between black-box and white-box accuracy is precision: the white-box attack gave no false positives, whereas black-box yielded about 15% in FiveThirtyEight and about 60% on GSS. This shows that the instance counts on tree paths gives useful information about the joint prior, allowing the attacker to better identify negative instances of the sensitive feature. Measured by both accuracy and precision, both attacks significantly outperform the random-guessing strategy, by at least 30%. However, neither attack achieves higher recall. This is due to the skewed prior distribution on the sensitive attribute (about 80% "No" in both cases), which leads the attack to favor answers which do not contribute to higher recall. The upshot of this condition is much higher precision, which supports greater confidence in positive predictions.

Figure 5a shows the advantage of the MI algorithms over the baseline strategy, with advantage defined as the increase in accuracy, precision, or recall. Both algorithms compare favorably in terms of precision on FiveThirtyEight (at least 80% improvement) and recall (at least 20% improvement), but are only slightly better in terms of accuracy (3-5% improvement). However, on GSS accuracy is slightly lower, whereas precision and recall are greater. Figure 5b shows the results as a percentage of the ideal strategy's performance. Reaching 100% of the ideal strategy's performance means that the attack performs as well as one could reasonably expect it to. The whitebox attack achieves this for precision, and comes close (within 15%) for accuracy.

Figure 5c compares the performance of the attack on instances from the training set versus instances outside it. Both attacks dramatically increase the attacker's chances of predicting the sensitive attribute, by as much as 70% precision and 20% recall. This demonstrates that inclusion in the training set poses a significant risk to the confidentiality of these individuals' responses.



(a) Results as advantage over baseline. (b) Results as a percentage of ideal. (c) Training vs. test attack performance.

Figure 5: BigML model inversion comparison to the baseline and ideal prediction strategies. Although the white-box attack achieved greater than 100% improvement over the ideal strategy on the FiveThirtyEight model, all percentages shown here are cropped at 100% to make the other results legible.

5. FACIAL RECOGNITION MODELS

Facial recognition models are functions that label an image containing a face with an identifier corresponding to the individual depicted in the image. These models are used increasingly often for a wide range of tasks such as authentication [8], subject identification in security and law enforcement settings [35], augmented reality [9], and photo organization (e.g., Facebook's "DeepFace" [1]). A growing number of web APIs support facial recognition, such as those offered by Lambda Labs [26], Kairos [21], and SkyBiometry [36]. A number of local libraries also expose APIs for facial recognition, such as OpenCV [5] and OpenBR [24]. Common to all these APIs is the ability to train a model using a set of images labeled with the names of individuals that appear in them, and the ability to perform classification given some previously trained model. Notice that classification may be performed by a larger set of individuals, including ones who do not have access to the labeled training set; use of the APIs in this way has been suggested, for example, in augmented reality applications [9].

In this section we discuss two MI attacks on the models used by these APIs to violate the privacy of subjects in the training set. Both attacks assume that the adversary has access only to a trained model, but not to any of the original training data.

1) In the first attack we assume an adversary who knows a label produced by the model, i.e. a person's name or unique identifier, and wishes to produce an image of the person associated with that label (i.e., the victim). The adversary uses MI to "reconstruct" an image of the victim's face from the label. This attack violates the privacy of an individual who is willing to provide images of themselves as training data, as the adversary can potentially reconstruct images of every individual in the training set. The adversary "wins" an instance of this attack if, when shown a set of face images including the victim, he can identify the victim. In subsequent text, we refer to this as the *reconstruction attack*.

2) In the second attack we assume an adversary who has an image containing a blurred-out face, and wishes to learn the identity of the corresponding individual. The adversary uses the blurred image as side information in a series of MI attacks, the output of which is a deblurred image of the subject. Assuming the original image was blurred to protect anonymity, this attack violates the privacy of the person in the image. Note that the image has been blurred to an extent that the model no longer classifies it correctly. The adversary can only hope to succeed if the individual was in the training set for the model. Here the adversary wins if he identifies the victim from a set of face images taken from the training set, or if the subject of the blurred image was *not* in the training set, and the adversary determines that the image produced by the attack does not correspond to any of the faces. We refer to this as the *deblurring attack*. This attack builds directly on the reconstruction attack. Due to space constraints we defer discussing it in detail, with experimental results, to the companion technical report [14].

5.1 Background

There are many proposed algorithms and models for performing facial recognition. Traditional approaches often relied on complex, hand-coded feature extraction, alignment, and adjustment algorithms, but more recently a number of promising systems have achieved better performance using neural networks [1,19,27]. These models are quickly becoming the standard by which facial recognition systems are evaluated, so we consider three types of neural network models: softmax regression, a multilayer perceptron network, and a stacked denoising autoencoder network. These models vary in complexity, with softmax regression being the simplest and the stacked denoising autoencoder network being the most complex.

Softmax regression. This classifier is a generalization of logistic regression that allows the class variable to take more than two values—in our case, there are 40 individuals in the dataset, so the classifier needs to distinguish between 40 labels. Softmax regression is often used as the final layer in deep neural network architectures, so on its own this classifier can be seen as a neural network with no hidden layers.

Multilayer perceptron network. We use a multilayer perceptron network with one hidden layer of 3000 sigmoid neurons (or *units*) and a softmax output layer. This classifier can be understood as performing softmax regression after first applying a non-linear transformation to the feature vector. The point of this transformation, which corresponds to the hidden layer, is to map the feature vector into a lower-dimensional space in which the classes are separable by the softmax output layer.

Stacked denoising autoencoder network. This classifier is an example of a deep architecture, and consists of two hidden layers and one softmax output layer. The two hidden layers, which have 1000 and 300 sigmoid units, are each instances of a *denoising autoencoder*. An autoencoder is a neural network that maps its feature vector to a latent rep-

Algorithm 1 Inversion attack for facial recognition models.

```
function MI-Face(label, α, β, γ, λ)
    c(x) ≝ 1 − f̃_label(x) + AuxTerm(x)
    x_0 ← 0
    for i ← 1 ... α do
        x_i ← Process(x_{i−1} − λ · ∇c(x_{i−1}))
        if c(x_i) ≥ max(c(x_{i−1}), ..., c(x_{i−β})) then
            break
        if c(x_i) ≤ γ then
            break
    return [arg min_{x_i}(c(x_i)), min_{x_i}(c(x_i))]
```

resentation (typically in a smaller space), and then maps it back (i.e., reconstructs) into the original feature space. Autoencoders are trained to minimize reconstruction error, so vectors in the latent space can be thought of as compressed encodings of the feature space that should decode well for instances in the training data. The hope with this architecture is that these encodings capture the main factors of variation in feature space (much like Principal Component Analysis), leading to greater separability for the softmax layer.

Dataset. We trained each type of model over the AT&T Laboratories Cambridge database of faces [2]. This set contains ten black-and-white images of 40 individuals in various lighting conditions, facial expressions, and details (e.g., glasses/no glasses), for a total of 400 images. We divided the images of each person into a training set (7 images) and a validation set (3 images), and trained each model using pylearn2's stochastic gradient descent algorithm [15] until the model's performance on the training set failed to improve after 100 iterations. The error rate for each model is given in Figure 6.

Model	Error
Softmax	7.5%
MLP	4.2%
DAE	3.3%

Figure 6: Model accuracy.

Basic MI attack. We now turn to inversion attacks against the models described above. The features that we will attempt to invert in this case are the full vector of pixel intensities that comprise an image, and each intensity corresponds to a floating-point value in the range $[0, 1]$. In all of the attacks we consider, we do not assume that the attacker knows exact values for any of the pixels in the vector he is trying to infer. These factors combine to make this type of inversion substantially different from the previous cases we consider, so these attacks require new techniques.

Assuming feature vectors with n components and m face classes, we model each facial recognition classifier as a function, $\tilde{f} : [0,1]^n \mapsto [0,1]^m$. Recall that the output of the model is a vector of probability values, with the ith component corresponding to the probability that the feature vector belongs to the ith class. We write $\tilde{f}_i(\mathbf{x})$ as shorthand for the ith component of the output.

We use *gradient descent* (GD) to minimize a cost function involving $\tilde{f}$ to perform model inversion in this setting. Gradient descent finds the local minima of a differentiable function by iteratively transforming a candidate solution towards the negative of the gradient at the candidate solution. Our algorithm is given by the function MI-Face in Algorithm 1. The algorithm first defines a cost function c in terms of the facial recognition model $\tilde{f}$ and a case-specific function AuxTerm, which uses any available auxiliary information to inform the cost function. We will describe an instantiation of AuxTerm when we discuss facial deblurring. MI-Face then applies gradient descent for up to α iterations, using gradient steps of size λ. After each step of gradient descent, the resulting feature vector is given to a post-processing function Process, which can perform various image manipulations such as denoising and sharpening, as necessary for a given attack. If the cost of the candidate fails to improve in β iterations, or if the cost is at least as great as γ, then descent terminates and the best candidate is returned.

MI-Face needs to compute the gradient of the cost function c, which in turn requires computing the gradient of the facial recognition model $\tilde{f}$. This means that $\tilde{f}$ must be differentiable for the attack to succeed. $\nabla\tilde{f}$ can be computed manually or automatically using symbolic techniques. Our experiments use symbolic facilities to implement the latter approach.

Algorithm 2 Processing function for stacked DAE.

```
function Process-DAE(x)
    encoder.Decode(x)
    x ← NLMeansDenoise(x)
    x ← Sharpen(x)
    return encoder.Encode(vecx)
```

Figure 7: Reconstruction without using Process-DAE (Algorithm 2) (left), with it (center), and the training set image (right).

5.2 Reconstruction Attack

The first specific attack that we consider, which we will refer to as Face-Rec, supposes that the adversary knows one of the labels output by the model and wishes to reconstruct an image depicting a recognizable face for the individual corresponding to the label. This attack is a fairly straightforward instantiation of MI-Face (Algorithm 1). The attacker has no auxiliary information aside from the target label, so we define $\text{AuxTerm}(\mathbf{x}) \stackrel{\text{def}}{=} 0$ for all $\mathbf{x}$. Our experiments set the parameters for MI-Face to: $\alpha = 5000, \beta = 100, \gamma = 0.99$, and $\lambda = 0.1$; we arrived at these values based on our experience running the algorithm on test data, and out of consideration for the resources needed to run a full set of experiments with these parameters.

In all cases except for the stacked DAE network, we set Process to be the identity function. For stacked DAE network, we use the function Process-DAE in Algorithm 2. Since the adversary can inspect each of the model's layers, he can isolate the two autoencoder layers. We configure the attack to generate candidate solutions in the latent space of

the first autoencoder layer, and at each iteration, PROCESS-DAE decodes the candidate into the original pixel space, applies a denoising filter followed by a sharpening filter, and re-encodes the resulting pixels into the latent space. We find that this processing removes a substantial amount of noise from the final reconstruction, while enhancing recognizable features and bringing relative pixel intensities closer to the images from the training set. An example is shown in Figure 7. The image on the left was reconstructed using the label of the individual from the right image without the use of PROCESS-DAE, while the center picture was reconstructed using this processing step.

Experiments. To evaluate the effectiveness of the attack, we ran it on each of the 40 labels in the AT&T Face Database, and asked Mechanical Turk workers to match the reconstructed image to one of five face images from the AT&T set, or to respond that the displayed image does not correspond to one of the five images. Each batch of experiments was run three times, with the same test images shown to workers in each run. This allowed us to study the consistency of responses across workers. The images shown to Turk workers were taken from the validation set, and thus were not part of the training data.

In 80% of the experiments, one of the five images contained the individual corresponding to the label used in the attack. As a control, 10% of the instances used a plain image from the data set rather than one produced by MI-FACE. This allowed us to gauge the baseline ability of the workers at matching faces from the training set. In all cases, the images not corresponding to the attack label were selected at random from the training set. Workers were paid $0.08 for each task that they completed, and given a $0.05 bonus if they answered the question correctly. We found that workers were usually able to provide a response in less than 40 seconds. They were allowed to complete at most three tasks for a given experiment. As a safeguard against random or careless responses, we only hired workers who have completed at least 1,000 jobs on Mechanical Turk and achieved at least a 95% approval rating.

5.2.1 Performance

We ran the attack for each model on an 8-core Xeon machine with 16G memory. The results are shown in Figure 8. Reconstructing faces out of the softmax model is very efficient, taking only 1.4 seconds on average and requiring 5.6 epochs (i.e., iterations) of gradient descent. MLP takes substantially longer, requiring about 21 minutes to complete and on the order of 3000 epochs of gradient descent. DAE requires less time (about 11 minutes) but a greater number of epochs. This is due to the fact that the search takes place in the latent feature space of the first autoencoder layer. Because this has fewer units than the visible layer of our MLP architecture, each epoch takes less time to complete.

algorithm	time (s)	epochs
Softmax	1.4	5.6
MLP	1298.7	3096.3
DAE	692.5	4728.5

Figure 8: Attack runtime.

5.2.2 Accuracy results

The main accuracy results are shown in Figure 9. In this figure, *overall* refers to all correct responses, i.e., the worker selected the image corresponding to the individual targeted in the attack when present, and otherwise selected "Not Present". *Identified* refers to instances where the targeted individual was displayed among the test images, and the worker identified the correct image. *Excluded* refers to instances where the targeted individual was *not* displayed, and the worker correctly responded "Not Present".

Figure 9a gives results averaged over all responses, whereas 9b only counts an instance as correct when a majority (at least two out of three) users responded correctly. In both cases, Softmax produced the best reconstructions, yielding 75% overall accuracy and up to an 87% identification rate. This is not surprising when one examines the reconstruction produced by the three algorithms, as shown in Figure 10. The reconstruction given by Softmax generally has sharp, recognizable features, whereas MLP produces a faint outline of these features, and DAE produces an image that is considerably more blurry. In all cases, the attack significantly outperforms, by at least a factor of two, randomly guessing from the six choices as this would give accuracy just under 20%.

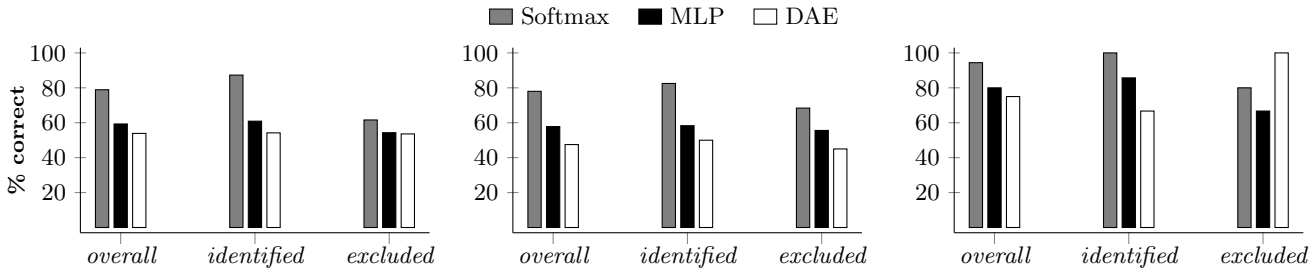
5.3 Black-Box Attacks

It may be possible in some cases to use numeric approximations for the gradient function in place of the explicit gradient computation used above. This would allow a black-box adversary for both types of attack on facial recognition models. We implemented this approach using scipy's numeric gradient approximation, and found that it worked well for Softmax models—the reconstructed images look identical to those produced using explicit gradients. Predictably, however, performance suffers. While Softmax only takes about a minute to complete on average, MLP and DAE models take significantly longer. Each numeric gradient approximation requires on the order of $2d$ black-box calls to the cost function, each of which takes approximately 70 milliseconds to complete. At this rate, a single MLP or DAE experiment would take 50–80 days to complete. Finding ways to optimize the attack using approximate gradients is interesting future work.

6. COUNTERMEASURES

In this section, we explore some potential avenues for developing effective countermeasures against these attacks. Ideally one would develop full-fledged countermeasures for which it can be argued that no future MI attack will succeed. Here we do a more limited preliminary investigation, showing discussing simple heuristics that prevent our current attacks and which might guide future countermeasure design.

Decision Trees. Just as the ordering of features in a decision tree is crucial for accuracy, it may also be relevant to the tree's susceptibility to inversion attacks. In particular, the level at which the sensitive feature occurs may affect the accuracy of the attack. To test this hypothesis, we implemented a variant of CART learning that takes a parameter ℓ which specifies the priority at which the sensitive feature is considered: the feature is only considered for splitting after $\ell - 1$ other features have already been selected, and removed from consideration afterwards. We then sampled 90% of the FiveThirtyEight dataset 100 times, and used this algorithm to train a decision tree on each sample for every value of ℓ. We evaluated the classification accuracy of the tree alongside white-box inversion performance.



(a) Average over all responses. (b) Correct by majority vote. (c) Accuracy with skilled workers.

Figure 9: Reconstruction attack results from Mechanical Turk surveys. "Skilled workers" are those who completed at least five MTurk tasks, achieving at least 75% accuracy.

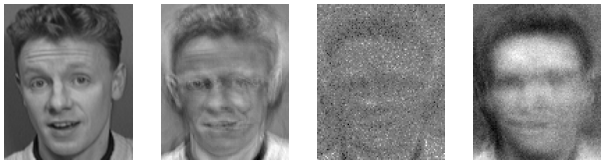


Figure 10: Reconstruction of the individual on the left by Softmax, MLP, and DAE.

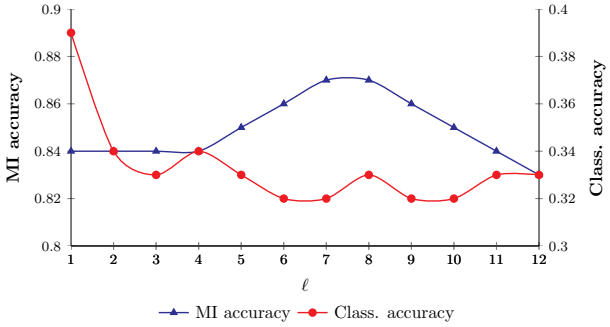


Figure 11: White-box MI vs. classification accuracy on decision trees trained on FiveThirtyEight data with the sensitive feature at each priority level ℓ. For this data, the optimal placement of the sensitive feature is at the first level, achieving the best classification accuracy while admitting MI accuracy only 1% greater than baseline.

The results are displayed in Figure 11 (we observed similar trends for black-box performance). The effectiveness of the attack in this case is clearly affected by the depth at which the sensitive feature appears in the tree. When the feature appears near the top or bottom of the tree, the attack fails with greater probability than otherwise. Furthermore, although prioritizing placement of the sensitive feature at a particular level does impact model accuracy, there is an optimal placement in this case: when the feature is placed at the top of the tree, classification accuracy is maximized while inversion accuracy is only 1% greater than baseline guessing. This suggests that it may be possible to design more sophisticated training algorithms that incorporate model inversion metrics into the splitting criteria in order to achieve resistance to attacks without unduly sacrificing accuracy.

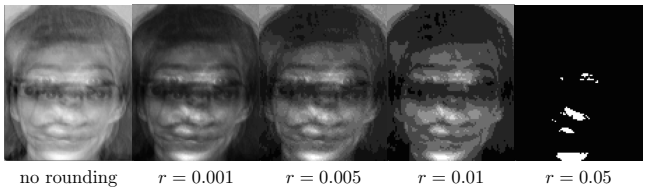


Figure 12: Black-box face reconstruction attack with rounding level r. The attack fails to produce a non-empy image at $r = 0.1$, thus showing that rounding yields a simple-but-effective countermeasure.

To understand why attack performance is not monotone in ℓ, we counted the number of times each tree used the sensitive feature as a split. This measure increases until it reaches its maximum at $\ell = 8$, and steadily decreases until $\ell = 12$. The difference in split frequency between $\ell = 8$ and $\ell = 12$ is approximately $6\times$. This is most likely because once most of the features have been used, the training algorithm deems further splitting unnecessary, thus omitting the sensitive feature from many subtrees. The inversion algorithm is unable to do better than baseline guessing for individuals matching paths through these subtrees, thus making the attack less effective.

Facial Recognition. Our attacks on facial recognition models are all based on gradient descent. One possible defense is to degrade the quality or precision of the gradient information retreivable from the model. There is no obvious way to achieve this in the white-box setting while preserving model utility, but in the black-box setting this might be achieved by reducing the precision at which confidence scores are reported. We tested this approach by rounding the score produced by the softmax model, and running the black-box reconstruction attack. The results are presented in Figure 12 for rounding levels $r = \{0.001, 0.005, 0.01, 0.05\}$; the attack failed to produce an image for $r = 0.1$. "No rounding" corresponds to using raw 64-bit floating-point scores to compute numeric gradients. Notice that even at $r = 0.05$, the attack fails to produce a recognizable image. This suggests that black-box facial recognition models can produce confidence scores that are useful for many purposes while remaining resistant to reconstruction attacks.

7. RELATED WORK

Machine-learning techniques are used in a variety of applications, such as intrusion detection, spam filtering, and virus detection. The basic idea behind these systems is to train a classifier that recognizes "normal behavior", so that malicious behaviors as can be labeled as abnormal. An active adversary can try to subvert these systems in a variety of ways, such as crafting malicious inputs that the classifier mistakenly labels as normal. This is called an *evasion* or *mimicry* attack. An adversary can also try to degrade the performance of such systems by devising inputs that create a large number of false alarms, thus overwhelming the system administrator. This is a variant of classic denial-of-service attacks. Barreno *et al.* [3] consider such attacks on machine learning algorithms. Lowd and Meek [30] extend these attacks to scenarios where the attacker does not have a complete description of the classifier. In contrast to this line of work, we target the privacy implications of exposing machine learning model functionality. However, a connection between these two lines of research will be interesting to explore in the future, as the mechanisms used for our attacks might prove useful in their settings.

Several authors have explored linear reconstruction attacks [10,12,23], which work as follows: given some released information $\mathbf{y}$ and assuming a hidden vector $\mathbf{s}$ of sensitive features (e.g., Boolean indicators for disease status of a group of patients), the attacker constructs a system of linear inequalities (a matrix A and a vector $\mathbf{z}$) such that $A\mathbf{s} \approx \mathbf{z}$, and attempts to solve for $\mathbf{s}$ using techniques that minimize some norm, such as LP decoding. Kasiviswanathan, Rudelson, and Smith [22] extended these attacks to releases that are non-linear, such as M-estimators. We also explore attacks on non-linear models, and we further investigate these attacks in realistic settings. It will be interesting future work to investigate whether strategies such as LP decoding work in settings similar to those considered in this paper.

Many disclosure attacks that have been explored in the literature. The classic attack by Sweeney [38] showed that it was possible to correlate publicly-available anonymzied hospital visit data with voter registration data, leading to re-identification of some individuals. Narayananan [32] demonstrated that an adversary with some prior knowledge can identify a subscriber's record in the anonymized Netflix prize dataset. Wang *et al.* [39], Sankararaman *et al.* [34], and Homer *et al.* [18] consider disclosure attacks on datasets generated from Genome-Wide Association Studies (GWAS) and other partial genetic information for an individual. Cormode showed that if an adversary is allowed to make queries that relate sensitive attributes to quasi-identifiers, then it is possible to build a differentially-private Naive Bayes classifier that accurately predicts a sensitive attribute [7]. Loukides *et al.* [29] show that one can infer sensitive patient information from de-identified clinical data. The main difference between this line of work and that presented in this paper is the type of information used by the adversary to infer sensitive information. Whereas previous work relied primarily on de-identified datasets, our attacks operate entirely on trained machine learning models and the metadata that is commonly released with them.

Komarova *et al.* [25] studied the problem of *partial disclosure*, where an adversary is given fixed statistical estimates from combined public and private sources, and attempts to infer the sensitive feature of an individual referenced in those sources. The key difference between theirs and our setting is that we assume the adversary is given a statistical estimator as a function, and can thus use it directly to make and evaluate predictions about individuals. It is worth studying whether the statistics used in their study can be used as additional side information to boost the performance of model inversion.

Li *et al.* [28] explore a privacy framework called *membership privacy*. This framework consists of two notions: positive membership privacy (PMP) and negative membership privacy (NMP). The framework gives rise to a number of security notions, including variants of differential privacy [11]. These security notions were not designed to resist MI attacks, and whether they might help guide countermeasure design is an open question. (See also the discussion of differential privacy in [13].) We also note that MI may serve as a primitive useful to adversaries seeking to violate these privacy notions (e.g., a high MI accuracy might reveal whether a person was in the training set).

8. CONCLUSION

We demonstrated how the confidence information returned by many machine learning ML classifiers enables new model inversion attacks that could lead to unexpected privacy issues. By evaluating our model inversion algorithms over decision trees published on a ML-as-a-service marketplace, we showed that they can be used to infer sensitive responses given by survey respondents with no false positives. Using a large-scale study on Mechanical Turk, we showed they they can also be used to extract images from facial recognition models that a large majority of skilled humans are able to consistently re-identify.

We explored some simple approaches that can be used to build effective countermeasures to our attacks, initiating an experimental evaluation of defensive strategies. Although these approaches do not constitute full-fledged private learning algorithms, they illustrate trends that can be used to guide future work towards more complete algorithms. Our future efforts will follow this path, as we continue to work towards new systems that are able to benefit from advances in machine learning *without* introducing vulnerabilities that lead to model inversion attacks.

9. REFERENCES

[1] DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[2] AT&T Laboratories Cambridge. The ORL database of faces. http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.

[3] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25. ACM, 2006.

[4] BigML. https://www.bigml.com/.

[5] G. Bradski. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, Jan. 2000.

[6] C.-L. Chi, W. Nick Street, J. G. Robinson, and M. A. Crawford. Individualized patient-centered lifestyle recommendations: An expert system for communicating patient specific cardiovascular risk

information and prioritizing lifestyle options. *J. of Biomedical Informatics*, 45(6):1164–1174, Dec. 2012.

[7] G. Cormode. Personal privacy vs population privacy: learning to attack anonymization. In *KDD*, 2011.

[8] M. Dabbah, W. Woo, and S. Dlay. Secure authentication for face recognition. In *IEEE Symposium on Computational Intelligence in Image and Signal Processing*, pages 121–126, April 2007.

[9] C. Dillow. Augmented identity app helps you identify strangers on the street. *Popular Science*, Feb. 23 2010.

[10] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, 2003.

[11] C. Dwork. Differential privacy. In *ICALP*. Springer, 2006.

[12] C. Dwork, F. McSherry, and K. Talwar. The price of privacy and the limits of lp decoding. In *STOC*, 2007.

[13] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *USENIX Security Symposium*, pages 17–32, 2014.

[14] Fredrikson, M. and Jha, S. and Ristenpart, T. Model inversion attacks and basic countermeasures (Technical Report). Technical report, 2015.

[15] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.

[16] Google. Prediction API. https://cloud.google.com/prediction/.

[17] W. Hickey. FiveThirtyEight.com DataLab: How americans like their steak. http://fivethirtyeight.com/datalab/how-americans-like-their-steak/, May 2014.

[18] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig. Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays. *PLOS Genetics*, 2008.

[19] G. Huang, H. Lee, and E. Learned-Miller. Learning hierarchical representations for face verification with convolutional deep belief networks. In *Computer Vision and Pattern Recognition (CVPR)*, June 2012.

[20] International Warfarin Pharmacogenetic Consortium. Estimation of the warfarin dose with clinical and pharmacogenetic data. *New England Journal of Medicine*, 360(8):753–764, 2009.

[21] Kairos AR, Inc. Facial recognition API. https://developer.kairos.com/docs.

[22] S. P. Kasiviswanathan, M. Rudelson, and A. Smith. The power of linear reconstruction attacks. In *SODA*, 2013.

[23] S. P. Kasiviswanathan, M. Rudelson, A. Smith, and J. Ullman. The price of privately releasing contingency tables and the spectra of random matrices with correlated rows. In *STOC*, 2010.

[24] J. Klontz, B. Klare, S. Klum, A. Jain, and M. Burge. Open source biometric recognition. In *IEEE International Conference on Biometrics: Theory, Applications and Systems*, pages 1–8, Sept 2013.

[25] T. Komarova, D. Nekipelov, and E. Yakovlev. *Estimation of Treatment Effects from Combined Data: Identification versus Data Security*. NBER volume Economics of Digitization: An Agenda, To appear.

[26] Lambda Labs. Facial recognition API. https://lambdal.com/face-recognition-api.

[27] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM.

[28] N. Li, W. Qardaji, D. Su, Y. Wu, and W. Yang. Membership privacy: A unifying framework for privacy definitions. In *Proceedings of ACM CCS*, 2013.

[29] G. Loukides, J. C. Denny, and B. Malin. The disclosure of diagnosis codes can breach research participants' privacy. *Journal of the American Medical Informatics Association*, 17(3):322–327, 2010.

[30] D. Lowd and C. Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647. ACM, 2005.

[31] Microsoft. Microsoft Azure Machine Learning.

[32] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, pages 111–125, 2008.

[33] J. Prince. Social science research on pornography. http://byuresearch.org/ssrp/downloads/GSShappiness.pdf.

[34] S. Sankararaman, G. Obozinski, M. I. Jordan, and E. Halperin. Genomic privacy and limits of individual detection in a pool. *Nature Genetics*, 41(9):965–967, 2009.

[35] C. Savage. Facial scanning is making gains in surveillance. *The New York Times*, Aug. 21 2013.

[36] SkyBiometry. Facial recognition API. https://www.skybiometry.com/Documentation#faces/recognize.

[37] T. W. Smith, P. Marsden, M. Hout, and J. Kim. General social surveys, 1972-2012. National Opinion Research Center [producer]; The Roper Center for Public Opinion Research, University of Connecticut [distributor], 2103.

[38] L. Sweeney. Simple demographics often identify people uniquely. 2000.

[39] R. Wang, Y. F. Li, X. Wang, H. Tang, and X. Zhou. Learning your identity and disease from research papers: information leaks in genome wide association studies. In *CCS*, 2009.

[40] Wise.io. http://www.wise.io/.