Noname manuscript No.
(will be inserted by the editor)

Beat the Machine: Challenging Humans to Find a Predictive Model's "Unknown Unknowns"

Josh Attenberg · Panagiotis Ipeirotis · Foster Provost

the date of receipt and acceptance should be inserted later

Abstract We present techniques for gathering data that expose errors of automatic predictive models. In certain common settings, traditional methods for evaluating predictive models tend to miss rare-but-important errors—most importantly, cases for which the model is confident of its prediction (but wrong). In this paper, we present a system that, in a game-like setting, asks humans to identify cases that will cause the predictive-model-based system to fail. Such techniques are valuable in discovering problematic cases that do not reveal themselves during the normal operation of the system, and may include cases that are rare but catastrophic. We describe the design of the system, including design iterations that did not quite work. In particular, the system incentivizes humans to provide examples that are difficult for the model to handle, by providing a reward proportional to the magnitude of the predictive model's error. The humans are asked to "*Beat the Machine*" and find cases where the automatic model ("*the Machine*") is wrong. Experiments show that the humans using Beat the Machine identify more errors than traditional techniques for discovering errors in from predictive models, and indeed, they identify many more errors where the machine is confident it is correct. Further, the cases the humans identify seem to be not simply outliers, but coherent areas missed completely by the model. Beat the machine identifies the "unknown unknowns." Beat the Machine is deployed at industrial scale. The main impact has been that firms are changing their perspective on and practice of evaluating predictive models.

> *"There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we know we don't know. But there are also unknown unknowns. There are things we don't know we don't know."*
>
> *– **Donald Rumsfeld***

1 Introduction

This paper discusses an important issue for machine learing researchers and practitioners to consider, that has received little if any attention in the machine learning literature, but becomes quite apparent when applying machine learning techniques in practice.

We worked with a firm to build systems for identifying web pages that contain instances of objectionable content, such as "hate speech" (e.g., racist content, antisemitism, and so on). The classification is based on models that take web pages as input and produces as output a score that can be used to block certain pages in the advertising ecosystem (or alternatively to produce reports on exposure). The firm would like to use this system to help protect advertisers, who (despite the best efforts of their advertising agents) sometimes see their ads appearing adjacent

Address(es) of author(s) should be given

to such objectionable content. The advertisers do not want their brands to be associated with such content, and they definitely do not want to support such content, explicitly or implicitly, with their ad dollars.

How does this firm assess the strengths and weaknesses of any system and model? Unfortunately for applying machine learning, there exists no representative, benchmark, "gold standard" corpus of hate speech, a trait common to many real-world machine learning problems. Traditional machine learning evaluation and training methods make an implicit closed-world assumption [9]. In logical systems, the closed-world assumption is that the only answers to a query Q are those that are actually in the database. In the context of predictive modeling, our closed-world assumption is that our labeled data are sufficient to give us a satisfactory estimation of model performance. Effectively, machine learning methods make the assumption that regularities that have no or insufficient representation in the training data essentially do not exist. Unfortunately, such an assumption is dangerously naive in applications with limited labeled training data, small disjuncts [13], and/or possibly unknown selection biases. In these cases, we face this problem: the model often cannot estimate properly its own performance in unseen data, and is vulnerable to the problem of *unknown unknowns*.

We will elaborate throughout the paper, but in a nutshell the problem is this: in ML research, we assume that the available data are representative and therefore we can use them safely to train and to evaluate our algorithms. Furthermore, and crucially, we assume that the models can estimate properly their own level of performance and report back accurate confidence metrics for different parts of the space. Assuming that we can know how well our algorithms are performing, we can work to improve the models, and/or improve the performance in regions of lower confidence in our predictions, and/or act based on this confidence. We have faith in our training data, and we focus on what we can know—including *knowing* what we don't know (e.g., regions of uncertainty in our learned model's behavior).

The problem is that, for various reasons, processes that produce training data can completely miss important regions of the space. Consider our case of identifying hate speech online: The category "hate speech" is relatively rare on the Web (fortunately) and extremely varied—a "disjunctive concept" in machine learning parlance. If a certain type of hate speech is not included in the training data, the resultant learned classifier is likely to (i) classify it as not-hate-speech, and worse, (ii) do so with high confidence. This is an "unknown unknown"—the classifier does not "know" this sort of hate speech, and it doesn't know that it doesn't know it. So, the "unknown unknowns" belong to the regions in the space where the model estimates the misclassification cost to be low, while in reality the misclassification cost is high.

This problem may be completely missed by standard ML evaluations: cross-validated error rates and AUCs for such models may look nearly perfect. However, the machine learning scientist would be incorrect to report to the application stakeholders that the models are essentially perfect. It is critical to emphasize that such evaluations only consider the "known unknowns." However, stakeholders of ML models need to also consider the possible impact of the unknown unknowns. For example, being blindsided by a client who discovers a completely unknown error, or suffering an embarassing PR disaster if the vulnerability of the model is exposed publicly by a third party or competitor. Finding the unknown unknowns, in a machine learning context, is what this paper is about.

We present a novel framework for thinking about errors of machine learning models that highlights the unknown unknowns (plenty of work has focused on the known unknowns). We then present a game-structured system (Beat the Machine) that takes advantage of crowdsourcing to help us identify the unknown unknowns. This system is fully implemented at industrial scale; we discuss several design iterations that each presented hurdles to overcome and led to the current system. Finally we present experimental results demonstrating that on real problems (including finding hate speech pages), explicitly focusing on finding unknown unknowns (with Beat the Machine) indeed finds pages that were completely missed by the prior system, are systematic errors (rather than just isolated outliers), and are systematically different from the prior training data.

2 Background and Scope

Many businesses and government organizations make decisions based on estimations made by explicit or implicit models of the world. Being based on models, the decisions are not perfect. Understanding the imperfections of the models is important (i) in order to improve the models (where possible), (ii) in order to prepare to deal with the decision-making errors, and (iii) in some cases in order to properly hedge the risks. However, a crucial challenge is that, for complicated decision-making scenarios, we often do not know where models of the world are imperfect and/or how the models' imperfections will impinge on decision making. *We don't know what we don't know.*

We see the results of such failures of omniscience in grand catastrophes, from terrorist attacks to unexpected nuclear disasters, in mid-range failures, like cybersecurity breaches, and in failures of operational models, such as predictive models for credit scoring, fraud detection, document classification, etc.

Specifically, this paper considers applications where:

- Every decision-making case can be represented by a description and a target. We have a (predictive) model that can give us an estimate or score for the target for any case. For this paper, we assume, for simplicity and without loss of generaliry, that the target is binary, and that the truth would not be in dispute if known.[1]
- We want to understand the inaccuracies of the model—specifically, the errors that it makes, and especially whether there are systematic patterns in the errors in regions of the space where the model is confident about its decisions and provides a very low estimate for misclassification costs. For example, is there a particular sort of hate speech that the model builders did not consider, and therefore the model misses it, while at the same time being confident about the reported decision?
- Finally, there are important classes or subclasses of cases that are very rare, but nevertheless very important. The rarity often is the very reason these cases were overlooked in the design of the system. In our example, hate speech on the web itself is quite rare (thankfully). Within hate speech, different subclasses are more or less rare. Expressions of racial hatred are more common than expressions of hatred toward dwarves or data miners (both real cases).

These problem characteristics combine to make it extremely difficult to discover system/model imperfections. Just running the system, in vitro or in vivo, does not uncover problems; as we do not observe the true value of the target, we cannot compare the target to the model's estimation or to the system's decision.

We *can* invest in acquiring data to help us uncover inaccuracies. For example, we can task humans to score random or selected subsets of cases. Unfortunately, this has two major drawbacks. First, due to the rarity of the class of interest (e.g., hate speech) it can be very costly to find very few positive examples, especially via random sampling of pages. For example, hate speech represents far less that 0.0001% of the population of (ad supported) web pages, with unusual or distinct forms of hate speech being far rarer still. Thus we would have to invest in labeling more than a million randomly selected web pages just to get one hate speech example, and as has been pointed out recently, often you need more than one label per page to get high-quality labeling [11, 8].

In practice, we often turn to particular heuristics to identify cases that can help to find the errors of our model. There has been a large amount of work studying "active learning" which attempts to find particularly informative examples [10]. A large number of these strategies (uncertainty sampling, sampling near the separating hyperplane, query-by-committee, query-by-bagging, and others) essentially do the same thing: they choose the cases where the model is least certain, and invest in human labels for these. This strategy makes sense, as this is where we would think to find errors. Additionally, there has been a long history of understanding that

[1] For our example, the description of the case would be the web page (its words, links, images, metadata, etc.). The target would be whether or not it contains hate speech.

"near misses" are the cases to use to best improve a model, both for machine learning [14] and for human learning [12].

Unfortunately, although helpful in understanding and improving modeling, for finding unknown unknowns, these strategies look exactly where we don't want to look. These strategies explicitly deal with the "known unknowns." The model is uncertain about these examples—we "know" that we don't know the answer for them (i.e., we have low confidence in the model's output). These strategies explicitly eschew, or in some cases probabilistically downweight, the cases that we are certain about, thereby *reducing* the chance that we are going to find the unknown unknowns.

In what follows, we next discuss changes to how we need to view the evaluation of classifiers, if we want to move from a closed-world view of a predictive modeling problem to an open-world view. Then we introduce a technique and system to use human workers to help find the *unknown unknowns*. Our BeatTheMachine (BTM) system combines a game-like setup with incentives designed to elicit cases where the model is confident and wrong. Specifically, BTM rewards workers that discover cases that cause the system to fail. The reward increases with the magnitude of the failure. This setting makes the system behave like a game, encouraging steady, accurate participation in the tasks. We describe our first experiences by the live deployment of this system, in a setting for identifying web pages with offensive content on the Internet. We show that this BTM setting discovers cases that are inherently different than the errors identified by a random sampling process. In fact, the two types of errors are very different. The BTM process identifies "big misses" and potential catastrophic failures, while traditional model-based example selection identifies "near misses" that are more appropriate for fine-tuning the system. The evidence shows that BTM does not just find individual "oddball" outlier cases, but it finds systematic big errors. In a sense, the BTM process indeed gives us the opportunity to learn our "unknown unknowns" and warn us about the failures that our current automatic model cannot (yet) identify by itself.

3 Unknown Unknowns

In order to provide a detailed discussion of unknown unknowns in the context of a predictive model, it is first necessary to formalize the concepts we will be discussing. Let x represent an example belonging to some problem space X. In classification settings, x has a "true" label, $\bar{y}$ from some set of possible labels Y. The task of a classification is to construct some predictive model, $f(x)$, that can estimate a label for each incoming example ($y = f(x)$) such that the estimated label y mirrors the (usually hidden) true label, $\bar{y}$, as closely as possible. There is a cost c_{ij} for a misclassification decision, when we classify an example from the true category $\bar{y} = y_i$ into a category $f(x) = y_j$. In this work, we are concerned only with models that output a posterior probability estimate over the set of available labels, that is, $f(x) = p(y|x) = \langle p_1, \ldots, p_n \rangle$; such models, through the probability estimates, effectively also report the estimated misclassification cost of each example:

$$\widehat{ExpCost}(x) = \sum_{i,j} p_i \cdot p_j \cdot c_{ij} \quad \widehat{MinCost}(x) = \min_j \sum_i p_i \cdot c_{ij} \tag{1}$$

Such probabilities can then be used to select a preferred label, for instance by choosing the y with the highest probability, or in the cost-sensitive setting, choosing the label with the least expected cost [5]. Note that the focus on models that produce probability estimates is without loss of generality— there exists a variety of techniques for transforming "hard-labeling" models into probability estimators (see, eg. [4]).

Notice that Equation 1 provides *estimates* of the misclassification cost. The *actual* misclassification cost can be different and depends on the accuracy of the model and the generated posterior probability estimates. The difference between the estimated and the actual misclassification costs gives birth to four different classification scenarios, as depicted in Figure 1.

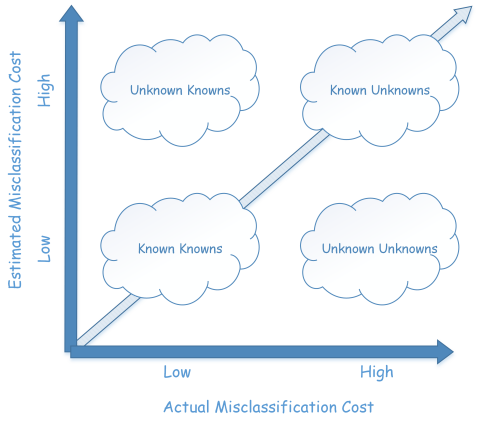


Fig. 1 The decisions made by a predictive model can be broadly separated into four conceptual regions: (i) The "known knowns," which are the examples for which the model is mostly correct and is also confident of being correct; (ii) The "known unknowns," which are the examples for which the model is often mistaken but also anticipates these mistakes by placing low confidence in the decisions; (iii) The "unknown knowns," which are the examples for which the model is often correct but returns very low levels of confidence; and (iv) The "unknown uknowns," which are the examples for which the model is wrong but also confident on being correct.

Along the diagonal, we have examples for which the estimates of the model in terms of misclassification cost are in line with the actual classification costs. In the lower-left corner we have the examples that the model can classify correctly *and* is confident about the reported decisions. These are the "*known knowns*". In the upper-right corner, we have the cases of "known unknowns": these are the cases where the model fails often but is also aware of this problem.

Definition 1 (Known Unknown) Let $X' \subset X$ be a region of the problem space and x be a randomly picked example from X'. We denote by $\widehat{ExpCost}(X')$ the expected misclassification cost of the examples in X', and $ExpCost(X')$ be the actual misclassification cost of the examples in X'. A *known unknown* is an example for which the actual misclassification cost $ExpCost(x)$ is high *and* the estimated misclassification cost $ExpCost(X')$ is also high.

Known unknowns as described in Definition 1 correspond to a commonly occurring notion in machine learning. The X' region corresponds to an "uncertainty" region, an area where the predictive model is unsure of itself, and where mistakes are likely. This concept has been exploited in a variety of contexts, for instance, when applied to the problem of gathering labels for the purpose of model training, selecting those examples within an ϵ-radius of the decision boundary corresponds to uncertainty sampling, perhaps the most well known active learning heuristic [7].

In the context of prediction-time classification, "known unknowns" are those cases for which errors are expected based on the confidence of the classification. These are cases where it may be less costly to "reject" than to make a risky label prediction. Classification with a "reject option" is an extension of traditional classification where in addition to labeling each example with some $y \in Y$, a predictive system may additionally defer prediction, either by ignoring an example entirely, or perhaps sending the example to a domain expert for manual evaluation [3,2]. Given that such "rejection" likely comes at some non-trivial cost $q(x)$, the task of classification with a reject option is then to balance the expected misclassification costs with the costs of rejection.

While it is important to understand the mistakes that your model is known to make and to react in an appropriate manner, models in production often make mistakes far from this area of predicted uncertainty. Consider the hate speech classification system discussed previously. While deployed in a production setting, this model is likely to encounter examples eliciting a high degree of predicted uncertainty.[2] Those managing the model can react to these borderline examples,

[2] For instance, a encyclopedia entry discussing racial issues or a history of racism.

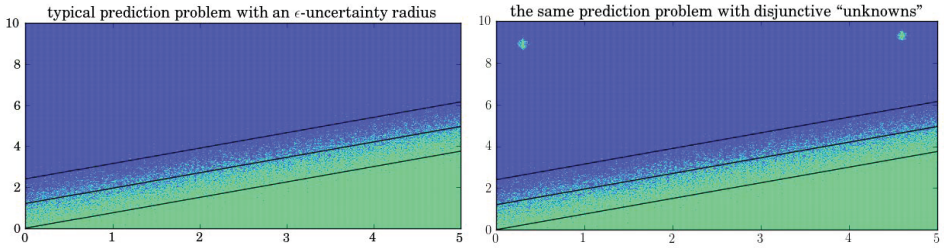


Fig. 2 A typical classification setting. On the the top we see the decision boundary that minimizes the prediction prediction error of a inseparable training set. Additionally, we see the ϵ-radius around the classifier where mistakes are though to occur. The bottom, we see the same classifier with the inclusion of small, disjunctive "unknowns", presenting mistakes that occur well outside a model's region of uncertainty.

and perhaps build some rough estimate of a model's overall exposure to misclassification risk. However, for a variety of reasons, the model may also encounter examples where it will assign a label with high confidence, and be wrong. Call all such examples "unknown unknowns."

Definition 2 (Unknown Unknown) Following Definition 1, an example x' is said to be an "*unknown unknown*" if x' is outside the region of uncertainty and has *low* estimated misclassification cost $\widehat{ExpCost}(X')$, but the actual misclassification cost $ExpCost(x)$ is high. In other words, the example is misclassified but the classifier is certain that it was correctly classified.

Definition 2 codifies the notion of an "unknown unknown". Intuitively, these are examples that are distant from any decision boundary, examples that the model is quite certain a correct label can be assigned, yet are still labeled incorrectly. While in the strict sense, the above definition includes "random noise"—individual examples that for whatever reason do not have the expected label[3]—the motivating case is disjunctive sub-regions of the problem space [13]. These are small, yet consistently labeled neighborhoods of examples isolated from the body of examples of the same class. These "islands" of examples may be sufficiently rare in the relative sense to avoid detection from random sampling processes used to generate training sets [1]. However, their absolute size and prevalence in many real world problems makes them a genuine risk.

Figure 2 presents a fairly typical classification scenario that might be impacted by "unknown unknowns". On the left, we see an inseparable two-class problem, with a linear decision boundary that minimizes the prediction errors on this space. Above and below this decision boundary, we see an a region of uncertainty, ϵ-wide, where mistakes are *known* to occur. This represents a typical post-training understanding of the problem space: data is gathered by some random process, for instance via active learning. An imperfect model is trained; however, areas where mistakes occur are known and expectations can be managed. On the right we see a typical scenario when such models are deployed in the wild—rare disjunctive sub-regions of the problem space emerge. These are portions of the problem space that escaped the initial sampling process used to generate the training set. These unknown unknowns while small in terms of their proportion of the problem space may still exist in large absolute numbers. However, because they are unobserved during model construction, they have likely escaped any possible contingency planning for dealing with their associated mistakes.

Finally, from Figure 1, we can see that there is one more type of example: the "*unknown knowns*". These are the cases where the model is incorrectly pessimistic: The model reports low confidence and high estimated cost for the examples in these region(s). However, in reality the model generates mostly correct predictions. Such cases are typically easier to manage but may still cause problems in the stakeholders: if this region generates many rejects (with an associated inspection and intervention costs), then the model will be perceived as being overly cautious and potentially inefficient. Despite the fact that it is a promising direction, identifying and dealing

[3] For instance, due to erroneous labeling, signal degradation, or non-pathological difficulties in data collection.

with cases of unknown knowns is beyond the scope of this paper, and we leave their study for future work.

4 Beat the Machine

Assessing the in-the-wild performance of any automated classification system can be challenge. Situations with class imbalance and rare disjunctive sub-concepts such as the hate speech classifier presented in Section 1 make accurate assessment particularly difficult and lead to the existence of unknown unknowns. Traditionally, we would sample from the output decisions and employ humans to verify the correctness of the classifications. Using these judgments we can estimate the error rate in different parts of the classification space. Unfortunately, given our problem characteristics, this process can be woefully inefficient. First, if the classification decisions are relatively accurate, then most of the results will be accurate, and without intelligent sampling, humans will encounter errors very infrequently. Second, if there is substantial class imbalance, most of the encountered errors would be misclassifications of examples truly of the majority class into the minority. This is problematic since in significantly imbalanced classification problems, the minority class generally incurs a far greater mistake cost—as in the case of hate speech, this is what is being predicted. Third, if the problem space has rare disjunctive sub-concepts, identification may be particularly tricky—chances of occurrence may be $1 : 1,000,000$ or less. In these situations, it can become quite difficult to identify misclassifications of examples truly in the minority class.

*Example 1 C*onsider the case of identifying pages with hate speech content. If we have a relatively accurate classifier, with 95% error rate on each class, it becomes very difficult to identify misclassified pages that contain hate speech. In a random sample, most of the pages are correctly classified as benign. Even in the unrealistically generous case that 0.1% of the pages on the Internet contain such objectionable content, to find one "false negative" (the severe error: hate speech passing as benign) we will have to inspect approximately $20,000$ pages (and in the process would find around $1,000$ false positives).
□

It is tempting to consider such problems inconsequential. However, when such a system is used to filter billions of pages, such "relatively infrequent" errors become frequent in absolute numbers. Furthermore, even "outlier" cases can cause significant damage, for example, to the public image of a company that accidentally supports a site containing such content through advertising. Unknown unknowns may be particularly damaging; client's expectations haven't been properly managed, and detailed contingencies are unlikely to exist.

Instead of passively waiting for such unknown errors to "emerge" we can instead actively seek to find them. We describe a system that engages human intelligence, accessed through crowdsourcing, to identify these "unknown unknown" cases. In a sense, this is similar to "white hat" hackers that are hired by companies to find vulnerabilities and break into their own security systems. In our case, human workers are asked to submit pages that will "beat" our classifier.

4.1 BTM Task Design

To describe the design of Beat the Machine, we now will walk through several designs of increasing sophistication, building up the ideas by focusing on challenges and subsequently more sophisticated designs.

Design 1: Let's start with a straightforward idea: Ask humans to find the unknown unknowns, i.e., the cases that "beat the machine"—the users would submit URLs that they believed would be incorrectly classified by the current classification model. To spur engagement, a user would receive a nominal payment for just submitting the URLs, and then she would receive a significant

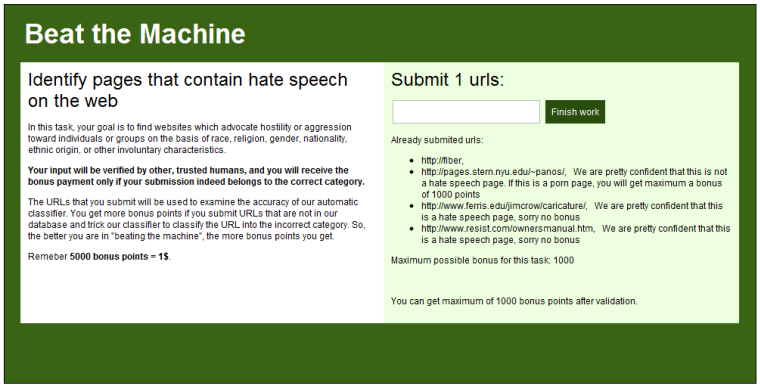


Fig. 3 A screen-shot of the BTM interface on Mechanical Turk.

bonus payment for every URL that was misclassified. (In the implementation, the nominal payment was 1 cent per 5 URLs, and the payment per misclassified URL was 50 cents.)

Of course there is an obvious problem: how could such a system tell that the URL indeed beats the machine. The whole point is to find cases that the system does not know that it gets wrong! To judge misclassification, we tasked another set of (trusted) humans to classify these URLs. Then, to determine whether the URL beat the machine, we compared the classification of the trusted set of humans with the outcome of the machine model. To avoid certain issues of gaming, the BTM workers were recruited through Amazon Mechanical Turk, and the trusted human judges were recruited and trained through oDesk for the fully automated system, and were student interns using a separate system for the experimental evaluation. Unfortunately, this simple design was not as effective as we would have liked, for a variety of reasons.

The first, and most obvious, problem that we encountered was the lack of interactivity. The workers could easily submit URLs that would break the model, but then they had to wait for other humans to inspect the results, in order to assess whether they had succeeded. This process would take from a few minutes to a few hours. The delay made the task opaque to the players of the BTM game, as they did not know if they were good in "playing the game" or not.

Adding immediate classification feedback: To resolve (partially) the lack of interactivity, we augmented the system to classify URLs on the fly, and give immediate feedback to the humans about the classifier outcome. (For example "The machine believes that this URL contains hate speech. Do you believe that this is correct?") The BTM player could then decide whether the URL was indeed a misclassification case and submit it for further consideration. Upon submission, the user received provisional bonus points that correspond to a cash reward. The bonus points became permanent (and the worker was paid) immediately after the inspection and verification of the submitted content by the human judges.

Unfortunately, this design did not provide the proper incentives. Players found it much easier to locate pages from the majority class (e.g., pages without any hate speech content) that would be misclassified as containing hate speech. So, instead of locating the desired, high-cost errors, we received the type of errors that we could find more easily by observing the positive classifications. (Recall that due to the class imbalance, most of the observed errors would be good pages being classified as containing hate speech.) As described above, we are particularly interested in finding pages that contain hate speech but are incorrectly classified as benign. (And especially, among these, the "unknown unknowns.") Furthermore, we experienced a significant number of cheating attempts where users were submitting random URLs and always insisting that the content is different than the classification decisions, even though the classifier was correct.

Segmenting the task by class: To deal with these problems, we split the task into two subtasks: (1) Seek pages in the minority class that are misclassified in the majority class (i.e., pages that contain offensive content but are classified as benign), and (2) seek pages with benign

content that would be classified as offensive. This segmentation simplified the overall design and made the task easier for participants to understand. Moreover, it allowed us to quickly reject submissions that were of no interest. For example, if we are asking for misclassified hate speech pages, we can quickly reject pages that our classifier unambiguously classifies as hate speech. (In the original design, users had the incentive to mark these as "non-hate-speech" hoping that the human judge would accept their judgments.) Figure 3 shows the (simple) task interface.

Expanding the incentives: In the final design , we also improved the incentive structure by rewarding differently users that discover "big mistakes" (the "unknown unknowns") and those that discover the "small mistakes" (the "known unknowns"). Instead of giving a constant bonus to the player for a misclassified URL, we reward misclassifications proportionally to the estimated misclassification cost, which we infer through the returned confidence. Examples that have high estimated misclassification cost are given a the reward is small. This was a known unknown. On the other hand, if the model is very confident in its decision (i.e., a classification confidence close to 100%) and correspondingly has a low estimated misclassification cost, but the decision is incorrect, then the BTM system gives the highest possible bonus to the worker.[4] If the confidence was lower, say 75%, the estimated misclassification cost was higher, and the reward was proportionally smaller. We also reward players that provide examples for which the model was correct but uncertain: if the model predicted that the page is 60% likely to contain hate speech, and the page indeed contained hate speech, the user received a small bonus.

5 Experimental Studies

In Section 3 we described the concept of the unknown unknowns and in Section 4 we described a gamified structure that incentivizes humans to identify such unknown unknowns in a predictive model.

To provide a experimental evaluation of BTM, we asked two questions:

- Does BTM identify errors efficiently?
- Does BTM identify isolated examples of unknown unknowns, or does it identify systematic unknown-unknown regions in the space?
- Can we use the discovered errors to improve the models?

For our experiments, we used the BTM system to challenge two classification systems. One for detecting pages with hate speech, and one for detecting pages with adult content. We ran the systems with the configuration details described in the previous section (0.2 cents for the base task, 50 cents maximum payment for a URL that generates an error).

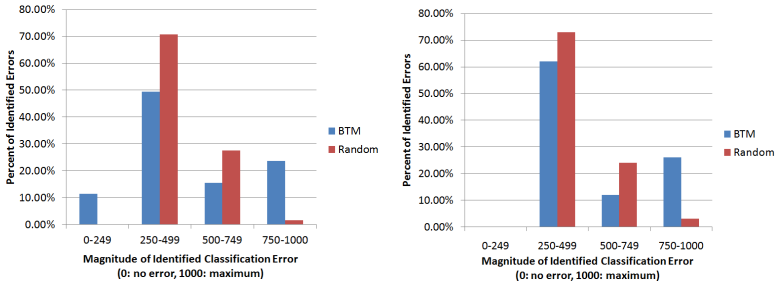
Comparison with stratified random examination:

In the application domain, the standard procedure for quality assurance of models such as these is stratified random examination of model predictions. Examining a uniform random sample of the output is particularly uninformative, as the classifiers are quite accurate and the distributions are quite unbalanced, and so the vast majority of cases are correctly classified and not objectionable. Therefore, standard procedure is to have human workers examine a random sample, stratified into bins by the model's confidence score. Specifically, the range of confidence scores [0,1] was divided into k equal-width bins. A set of N URLs for testing was sampled randomly, with $\frac{N}{k}$ from each bin. This stratification is used because it generally finds more errors—it over-samples the URLs for which the models have low confidence (and are likely to be wrong).[5] However, the discovered errors are likely to be "known unknowns."

We compared this quality assurance procedure with BTM. It is important not to see this as a bake-off. Although we will compare identified error rates, the bottom line is that the two procedures are complementary: they are designed to assess different quantities.

[4] In our particular implementation, the highest bonus is worth 1000 points, or 50 cents.

[5] It also allows the managment and data science teams to estimate the calibration of the scoring system, by examining the percentages of positive and negative instances in each bin.



(a) Hate Speech (b) Adult Content

Fig. 4 Distributions of the magnitude of the identified mistakes in the predictive model's output by BTM and by random sampling for two ad safety tasks. Each bar indicates the percentage of successfully identified mistakes that reside in the associated score range.

For the adult classifier, the human workers in the stratified examination identified errors in 16% of the inspected cases—*orders of magnitude* higher than the natural error rate of the classifier. In contrast, using BTM, more than 25% of the submitted cases exhibited an error. The corresponding statistics for hate speech favored BTM even more strongly: workers identified errors in only 9% of the inspections for stratified examination. Workers identified errors in 27% of the inspected URLs with BTM. On the surface, these results seem to indicate that the BTM process is more efficient than the standard quality assurance procedure in identifying problematic cases. However, you may have noted that we could increase the "efficiency" of the non-BTM procedure by simply sampling a larger proportion from the low-confidence predictions. Unfortunately, this would directly reduce the number of "unknown unknowns" discovered. At the extreme, the largest number of errors would be found by sampling only in the low-confidence region. All the errors found would then be known unknowns.

Comparing the severity of errors: Figure s4(a) and 4(b) show the distribution of modeling mistakes identified for the hate speech and adult content tasks, respectively. The blue bars show the mistakes identified by BTM; the red bars show the mistakes identified by the stratified evaluation. The number ranges on the horizontal axis show the severity of the errors, in four buckets—how badly the classifier misjudged its classification. The severities range from the least severe on the left (zero is no error), to maximum severity on the right: 1000 means that the classifier was certain of one class and the actual class was the other. The unknown unknowns are on the far right of each graph.

A consistent behavior is observed for both categories: a significantly larger proportion of the BTM errors found are severe misses—the unknown unknowns. Within the errors identified by BTM, 25% were cases of high severity; here the model was confident that it was making the correct decision (classifying the content as benign, with the highest confidence), but in reality the decision was incorrect.

In sum, only does BTM identify a larger number of problematic predictions than the stratified testing, but also a significant number of these predictions were unknown unknowns. These cases would be missed in practice and without an unpleasant event (possibly a catastrophe), we never would know that we had missed them. In contrast, and by now as expected, most of the identified errors for the stratified examination were misses that occur near the decision boundary.

Isolated outliers or systematic regularities? A natural question to ask is if the cases found by BTM seem to be isolated outliers, or whether they seem to be regularities. We framed this question pragmatically, based on the assumption that regularities can be modeled.[6]

Following this reasoning, we ran the following experiment. We attempted to learn a model that would classify positive and negative examples from amongst the BTM-identified cases. Internal consistency in the identified errors would suggest that these cases are not outliers, but

[6] So therefore we may miss regularities that fall outside the limits of the inductive bias of our modeling procedures.

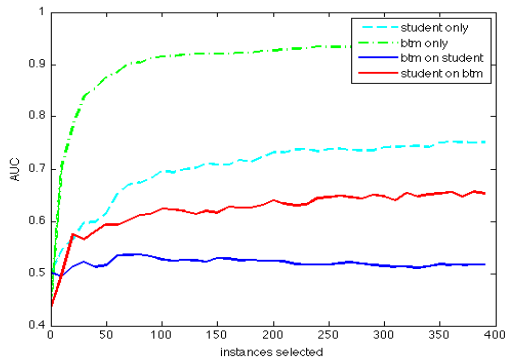


Fig. 5 Learning curves generated by the models using cross-validation (BTM and student lines), and then use as test case for BTM the errors identified by random sampling (BTM on students), and vice versa (students on BTM).

rather that they constitute parts of the space where the model fails systematically (potentially without being aware of the failures). Figure 5 shows the results of this process.

First consider the "btm only" learning curve and the "student only" learning curve, showing 10-fold cross-validated areas under the ROC curve (AUC). The "btm only" curve shows the performance of models built and tested using the error cases identified by the BTM process. The "student only" curve shows the performance of models built and tested using examples gathered through stratified examination (the pages selected by stratified examination were inspected by students for this experiment, hence the name). Importantly, the results show that the quality of both models is fairly high. This illustrates that there is consistency and internal coherence in these sets of identifed errors. The fact that the BTM model can reach high levels of accuracy indicates that BTM indeed identifies systematic regions that contain unknown unknowns, and not just disparate outliers. However, note the difference between the quality that is achievable by training with the two different data sets. The comparatively lower quality of the stratified examination model illustrates that these pages are inherently more difficult to learn from; this is consistent with our discussion above that the discovery via stratified random examination (DVSRE) focuses on the ambiguous cases (those that the current model is uncertain about), while BTM discovers incorrectly classified areas of the space that have been systematically ignored.

We also can examine whether the two approaches (DVSRE and BTM) identify sets of similar examples, or whether each of them identifies something completely different. For that, we tested the performance of BTM-trained models using the examples from DVSRE ("student") and vice versa. The results indicate that there is little cross-consistency between the models. What we discover using BTM has little effectiveness for classifying the error cases identified through DVSRE, and vice versa. This finding indicates that BTM and DVSRE reveals errors in different parts of the space; importantly, BTM finds errors that are systematically different from those found by DVSRE. BTM and DVSRE are different processes, capable of identifying different types of errors. Each of these has its place in the evaluation and improvement of automatic models. DVSRE identifies primarily cases where the model already knows that it is not confident. The results show that even if the DVSRE were stratified only on the "unknown" region, it still would not identify nearly as many unknown unknowns as Beat the Machine.

6 Impact in Industrial Deployments

The Beat the Machine design has directly changed the way several companies view and practice the evaluation of predictive systems. In our example domain for this paper, one medium-sized company that does massive-scale webpage classification in the online advertising space

has decided to move beyond its traditional system evaluation methods. Traditionally, before deployment models have been evaluated using cross-validation, expert model examination, and stratified case examination. Once a prediction system was deployed in practice, evaluation was based on continual stratified examination from the production classification stream. Let's call this practice "passive testing." According to the firm's founding data scientist, this work convinced this company that Beat the Machine and other "active testing" practices are vital to understand their predictive models' performance and alert stakeholders about areas of concern. The most convincing impact is that this firm has invested in the industrial development of Beat the Machine, and is pursuing its use across classification tasks (not just objectionable content).

Beat the Machine also has directly influenced the workflow design of a large firm that runs one of the most popular online labor marketplaces. Now, any automatic algorithmic system that gets deployed is tested by asking users to find cases that will break the system. For example, to test a job classification engine, users are asked to submit job descriptions that are legitimately and unambiguously classified into one category, but which the automatic system will classify into another. One of the interesting side-effects of this practice is that it catches early shifts in the content of the typical job posting. So, when a new type of task starts emerging in the market (and it cannot be classified properly by the current automated engine) the BTM system is likely to catch this trend early, before it becomes a major issue of user dissatisfaction.

Finally, the BTM system has been deployed as part of an image tagging service for a third company. According to one of the founders, before BTM, automatic systems together with humans were used to tag images with keywords. Under the new BMT-style design, there is an extra phase where humans are looking at an image to "challege" existing tags, with the goal that the newly provided tag will be better and more relevant than the currently assigned one. This design allows for higher quality keywords to be assigned to the images, avoiding cases where only a set of generic, uninteresting keywords are assigned to an image (either by algorithms or by humans).

In general, the main practical impact of the BTM system is the new approach for testing and debugging automatic machine learning models. The technique of rewarding users for locating vulnerabilities and bugs is common in security, and conceptually similar to BMT. However, there is a difference: When dealing with statistical models, merely locating an incorrect classification decision is hardly an event worth rewarding, or even recording. Our BTM system is designed to reward cases where the model exhibits a systematic failure, about which we are not aware until it happens.

7 Conclusion and Future Work

We presented the problem of "unknown unknowns" in the setting of predictive modeling and explored the design of the *Beat the Machine* process for directly integrating humans into testing automatic decision models for severe vulnerabilities. Our results suggest that BTM is especially good in identifying cases where the model fails, while being confident that it is correct. Several companies have implemented systems based on the ideas of Beat the Machine, after having read initial reports of this research.

We believe that machine learning research should devote more study to this sort of system. Presumably, even though we have gone through several design iterations, there is a lot to learn about how to design such systems to work well. As we discussed in the deployment section, in addition to using BTM proper, companies already have been using the ideas in ways slightly different from the exact design we've presented. Furture, it is naturally interesting to ask how to best use knowledge of such vulnerabilities to improve the automatic decisions models. To our knowledge, no work has yet studied this. Our preliminary experiments indicate that building predictive models in the BTM setting is a very complicated problem. For example, oversampling cases where a model makes big mistakes can be catastrophic for learning (think simply about oversampling outliers in a linear regression). On the other hand, techniques like boosting [6] have gotten tremendous advantage by overweighting cases where the current model is incorrect.

The potential benefit of being able to simultaneously explore a model's unknowns and offer robust model improvement would be an exciting advance.

References

1. J. Attenberg and F. Provost. Inactive Learning? Difficulties Employing Active Learning in Practice. *SIGKDD Explorations*, 12(2):36–41, 2010.
2. C. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, January 1970.
3. C. K. Chow. An Optimum Character Recognition System Using Decision Functions. *IEEE Transactions on Electronic Computers*, (4):247–254, December 1957.
4. P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *KDD '99*, 1999.
5. C. Elkan. The Foundations of Cost-Sensitive Learning. In *IJCAI*, pages 973–978, 2001.
6. Y. Freund and R. E. Schapire. A short introduction to boosting, 1999.
7. D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *SIGIR '94*, pages 3–12, New York, NY, USA, 1994. Springer-Verlag New York, Inc.
8. V. Raykar, S. Yu, L. Zhao, A. Jerebko, C. Florin, G. Valadez, L. Bogoni, and L. Moy. Supervised Learning from Multiple Experts: Whom to trust when everyone lies a bit. In *ICML*, pages 889–896. ACM, 2009.
9. R. Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977.
10. B. Settles. Active learning literature survey, 2010.
11. V. S. Sheng, F. Provost, and P. G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *KDD '08*, 2008.
12. K. VanLehn. Analogy events: How examples are used during problem solving. *Cognitive Science*, 22(3):347–388, 1998.
13. G. M. Weiss. The impact of small disjuncts on classifier learning. In R. Stahlbock, S. F. Crone, and S. Lessmann, editors, *Data Mining*, volume 8, chapter 9, pages 193–226. Springer US, Boston, MA, 2010.
14. P. Winston. Learning structural descriptions from examples. 1970.