Debugging Machine Learning Extended Abstract

Gabriel Cadamuro[a]
Microsoft Research
Redmond, WA 98052, USA
cadamuro.gabriel@gmail.com

Ran Gilad-Bachrach
Microsoft Research
Redmond, WA 98052, USA
rang@microsoft.com

Xiaojin Zhu[b]
Microsoft Research
Redmond, WA 98052, USA
jerryzhu@cs.wisc.edu

[a]Current address: University of Washington, Seattle, WA, 98195, USA
[b]Current address: University of Wisconsin–Madison, WI, 53706, USA

Abstract

Creating a machine learning solution for a real world problem often requires multiple iterations of investigation and improvement until it reaches satisfactory performance. Even after deployment, it is common to discover limitations of the model or changes in the target concept that necessitate modifications to the training data and parameters. However, as of today, there is no common wisdom about what these iterations consist of, nor what debugging tools are needed to aid the investigative process. In this work we present a novel technique to help model developers find the root causes of prediction error on test items (henceforth 'bugs') and so help the developer to fix them. Given an observed bug our method aims to identify the training items most responsible for biasing the model towards giving the wrong prediction on the specific test item. This set of training items can aid in discovery of common errors like faulty training labels or poor training data coverage. Our method is applicable over many different learners, including deep neural nets with large and complex model representations, as well as many different data types.

Author Keywords

Machine Learning; Debugging

ACM Classification Keywords

I.5.1 [Computing Methodologies]: PATTERN RECOGNITION—*Models*; D.2.5 [Software]: SOFTWARE ENGINEERING—*Testing and Debugging*

Introduction

While the application of Machine Learning techniques in industry during the age of Big Data has led to models of incredible accuracy and practical use, the resulting systems have become correspondingly unwieldy and difficult to manage. As is the case in almost any engineering discipline, the process of developing a machine learning model is an iterative process such that in each iteration previous models are tested, problems are discovered, the root cause for the problems are identified and an improved model is developed. Therefore, each iteration includes testing, debugging and development. However, as of today, there are no common debugging tools for machine learning and very little research have been conducted trying to come up with such tools.

It is important to distinguish between debugging the learning algorithms and debugging the model [6]. In this work we focus on the later. That is, we assume that the learning algorithm is correct; however, the model that was generated makes wrong predictions. Therefore, imperfections in the model could be a result of the information provided in the training data – for example, mislabeled items and missing features. The goal of this study is to suggest techniques that allow the developer to identify the root cause of incorrectly predicted test items and to help in proposing remedies.

A case study: debugging a spell corrector

To motivate this work, we give a real life example that shows the impact that developing machine learning debugging

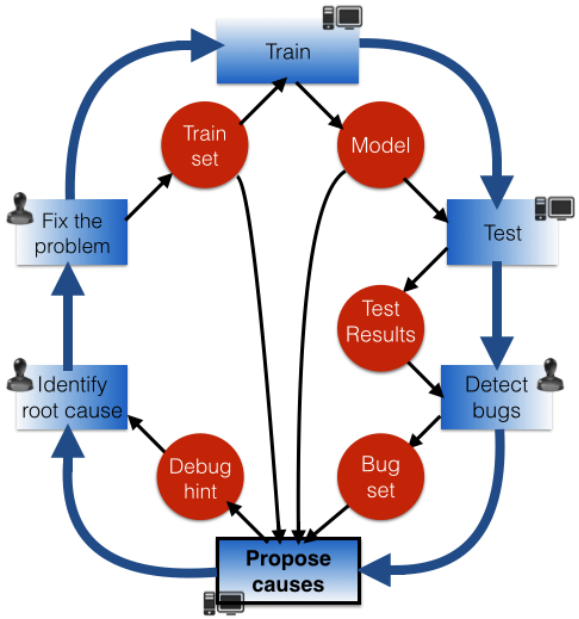

Figure 1: Diagram representing a generic machine learning debugging workflow on the outside (each square represents a discrete process) and the data used by this workflow as circles in the center. Prerequisite data for a process is shown by an incoming arrow into a process while the output of a process is linked by an outgoing arrow. The process are also identified by icons showing whether they are performed entirely by algorithms or with at least a degree of human intervention. For briefness, process can be referenced by their verb: for example the "identify root cause" process will be referred to as the identification process. This study focuses on the "Propose cause" process that provides a debug hint to the developer to help them find the root cause of a bug.

tools can provide. This example came about while collaborating with a group of developers and data scientists who were responsible for the spell correction component in the Bing search engine. This component receives the query the user sent together with a proposed correction and predicts the quality of this correction. In other words, the speller component is a classifier whose input is x =(query, correction) and whose output y is a quality label among "Perfect, . . ., Bad". For example, for the input query-correction pair (rnold schwarazenegger, arnold schwarzenegger) we expect this component to predict y =Perfect since "arnold schwarzenegger" is a good spell correction to the query "rnold schwarazenegger".

Having recently re-trained their spell correction model with new sources of training data, the spell corrector team found that while the new speller model had higher overall accuracy it also made some new embarrassing mistakes. One of these new mistakes was that the speller gave a high score to the test item (Portland or, Portland). That is, this model suggested that the query "Portland or" should be converted to the query "Portland". Note, however, that there are several cities in the US with the name "Portland" while "or" is state code of Oregon and therefore, the intent of the user was to query for the city of Portland in Oregon. However, if the speller would have dropped the "or" from the query, the search engine would return results on other cities, for example Portland in the state of Maine.

The question though is, why did the model find that this spell correction is good? are there similar problems that were not identified? How can this bug be fixed? Given the size of the model (hundreds or thousands of trees) and the size of the training data (millions of items), manual inspection of either of them was not a feasible solution.

To help mitigate such problems, we proposed a technique that scanned the entire training data and tried to identify a small subset of the training data that had the largest influence on making the model return this anomalously high score for the test item. The output of our technique on the test item (Portland or, Portland) is illustrated in Table 1. In this case the technique returned a subset of the training data that was helpful to identify the root cause of the bug. Specifically, the training items (Dayton-TN, Dayton), (Cascadia,OR, cascadia), (F-statistics, statistics) all have training label "Perfect". This is wrong because the corrections are in fact undesirable from a spell checking perspective. Our selected subset of training items thus enabled the developers to identify the root cause: it turns out there was an error in how labels were applied to a specific training file. Upon fixing it and re-running the training process, the developers corrected this test error and improved the overall performance of the spell checking model.

training x =(query, correction)	label y	influence
(Dayton-TN, Dayton)	Perfect	0.958
(waterloo wi, waterloo)	Bad	0.957
(humor.on, humor)	Bad	0.940
(Cascadia,OR, cascadia)	Perfect	0.931
(F-statistics, statistics)	Perfect	0.928

Table 1: A snapshot showing the output of our debugging tool for the bug on test item x =(Portland or, Portland). The "influence" column is the score produced by our technique to indicate how important this training data item was to the bug in question (with 1 being the highest score). Those highlighted in red indicate training labels later deemed incorrect by developers.

The bigger picture

As discussed earlier, developing and maintaining a machine learning solution is a process in which the developer tries to improve the solution. A schematic view of this process

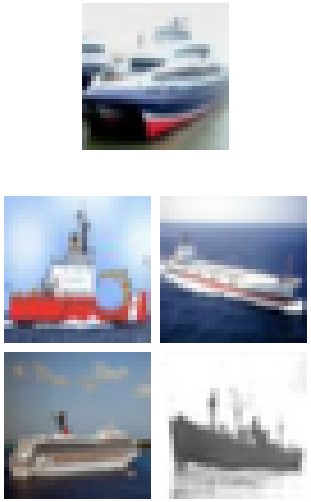

Figure 2: A demonstration of our technique on a bug in a deep–neural–network trained for image classification (CIFAR). If the top test image was misclassified, the bottom four training images are a possible debug hint in the sense that changing their training labels is likely to change the prediction on the top image. Note that the images are not pixel-wise similar.

is presented in Figure 1. The focus of this work is in assisting the developer who identified a bug (or a set of bugs) in finding the root cause. In order to provide a generic solution that works for different data types and learning algorithms, we propose the following approach. Given the bug, our technique scans the training data and finds a (small) set of training items such that if the labels of these items are modified, a model trained on the modified data would not have the bug. This small set of training items and their original labels, which we call the "debug hint", can be presented to the developer to help finding the root cause of the bug.

The information presented in the debug hint is a subset of the training data. Therefore, it has the same type as the data the developer is using and thus can comprehend. Our main technical contribution is to formulate the problem of finding a good debug hint as an optimization problem. Although this optimization problem is model dependent (neural-networks, trees, Gaussian-processes,...), the developer does not have to understand the internals of the algorithm being used. In our spell checker case study we demonstrated how the debug hint has been used to find a data source containing wrong labels. As another example, Figure 2 demonstrate an image classification task that was trained with deep-neural-networks. A developer does not have to understand the way neural-networks work in order to use the debug hint.

Previous work

There has been some prior work on debugging machine learning models within which we wish to frame our work. We do so by considering them in the context of Figure 1 to give us a consistent taxonomy in which to consider where these projects focused their attention and where their achievements lie.

The closest work to the one presented here is the work of Kulesza et al. [6] which discusses explicitly the problem of debugging machine learning models which they refer to as machine learning programs. They introduced the idea of providing explanations that the developer can use to debug: thus considering different version of the proposal process while empirically evaluating performance by the result of the fixing process. In contrast to our work, they focus on specific type of data (natural language) and assume that the model that was learn can be expressed as rules that are human intelligible.

Brooks et al. [3] presented a tool to assist developers in the ideation process of new features in the context of text base models. They proposed a visualization tool that highlights mislabeled examples to allow the user to think about new features. Again, this system is limited to specific type of problems (NLP).

Following Fails and Olsen Jr [4] there have been studies on interactive machine learning (see, for example, [2, 1]). These studies look at interfaces between humans and machines that allow for very rapid development iterations such as in recommender systems in which every piece of information provided by the user has an immediate effect on the model. They argue that this interaction should allow the user to gain insight about the underlying logic of the model to allow the user to provide the right feedback to steer it in the right direction [1]. The main focus of this line of research is on the interface design while our focus is on algorithms that find good 'hints'.

Kulesza et al. [5] showed that when users have a better understanding of the model, they provide better feedback to improve the model. The users in this study were not machine learning developers but users of a music recommendation system. Nevertheless, they managed to achieve their goals better and correct the system when it

was making mistakes, showing that when the identification process is well designed (for example, when identifying the root cause of why a music recommender presented a poor song choice, users have options like "song is too slow") useful debugging is not restricted to engineers and experts. Rosenthal and Dey [7] studied ways to interact with user to solicit correct labels and thus prevent bugs. However, they do not discuss how to handle bugs (the fixing process) or how to discover which unimplemented features should be added next.

Our contribution

The main contribution of this work is a method that allows debugging machine learning solution which is not restricted to specific type of data or specific algorithms. We propose a debugger that presents a subset of the training data as the debug hint. This explanatory subset of training data should be the data most responsible for creating the bug during training time: namely if it was removed or altered the bug would be less likely to exist or less severe; we also want the debug hint to be reasonably small so as to be comprehensible to a human developer. These two competing criteria thus define a notion of optimality for the debug hint set: the optimal debug hint should be both highly responsible and small in size at the same time.

We imagine several ways in which common root causes could be detected via such debug hint. Mislabeled training data can be identified by consistently showing up in these subsets as we saw in our case study. Low coverage for specific data items might be identified by bugs which return very small explanatory subsets while feature blindness (when a critical feature for this learning task has not yet been implemented) might be identified by finding that the explanatory subset that is proposed contains examples that are not expected to be similar.

Our work provides a formulation of the machine debugging flow and its main processes. The core of our formulation is a “debug hint optimization” problem to identify the optimal debug hint. This optimization problem is in general a multi-objective optimization problem. For certain learners, such as ordinary least squares and Gaussian process regression, we exhibit closed-form solution for finding the Pareto optimal solutions. For other learners such as ensembles of trees and neural networks, we propose an approximation technique to the optimal debug hint set and then test its performance empirically.

We hope that this work will encourage other researchers to make further contributions to the study of the entire life cycle of machine learning based engineering.

References

[1] Saleema Amershi, Maya Cakmak, W Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *AI Magazine* 35, 4 (2014), 105–120.

[2] Saleema Amershi, James Fogarty, Ashish Kapoor, and Desney S Tan. 2011. Effective End-User Interaction with Machine Learning.. In *AAAI*.

[3] Michael Brooks, Saleema Amershi, Bongshin Lee, Steven M Drucker, Ashish Kapoor, and Patrice Simard. 2015. FeatureInsight: Visual Support for Error-Driven Feature Ideation in Text Classification. In *IEEE Symposium on Visual Analytics Science and Technology (VAST)*.

[4] Jerry Alan Fails and Dan R Olsen Jr. 2003. Interactive machine learning. In *Proceedings of the 8th international conference on Intelligent user interfaces*. ACM, 39–45.

[5] Todd Kulesza, Simone Stumpf, Margaret Burnett, and Irwin Kwan. 2012. Tell me more?: the effects of men-

tal model soundness on personalizing an intelligent agent. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1–10.

[6] Todd Kulesza, Simone Stumpf, Margaret Burnett, Weng-Keen Wong, Yann Riche, Travis Moore, Ian Oberst, Amber Shinsel, and Kevin McIntosh. 2010. Explanatory debugging: Supporting end-user debugging of machine-learned programs. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*. IEEE, 41–48.

[7] Stephanie L Rosenthal and Anind K Dey. 2010. Towards maximizing the accuracy of human-labeled sensor data. In *Proceedings of the 15th international conference on Intelligent user interfaces*. ACM, 259–268.