Debugging Machine Learning Models

Gabriel Cadamuro GABCA@CS.WASHINGTON.EDU
Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2350, USA

Ran Gilad-Bachrach RANG@MICROSOFT.COM
Microsoft Research Redmond, WA 98052, USA

Xiaojin (Jerry) Zhu JERRYZHU@CS.WISC.EDU
Microsoft Research Redmond, WA 98052, USA

Abstract

Creating a machine learning solution for a real world problem often becomes an iterative process of training, evaluation and improvement where the best practices and generic solutions are few and far between. Our work presents a novel solution for an essential step of this cycle: the process of understanding the root causes of ’bugs’ (particularly consequential or confusing test errors) discovered during evaluation. Given an observed bug, our method aims to identify the training items most responsible for biasing the model towards creating this error. We develop a optimization based framework for generating this information which leads to our method not only having simple analytic solutions for certain learners but to it also being applicable to any supervised learner or data type.

1. Introduction

While the application of Machine Learning techniques in industry during the age of Big Data has led to models of incredible accuracy and practicality, 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 where in each iteration previous models are tested, problems are discovered, root causes identified and an improved model is developed. Therefore, each iteration includes testing, debugging and development. However, as of today, there are few debugging tools for machine learning.

In this work we focus on one step of this iterative cycle, assisting the developer who identified a bug (where a bug is a particularly damaging or inexplicable test error) in finding the root cause. In order to provide a generic solution that works for different data types and learning algorithms, we focus on suspected root cause in the training data itself. Given the bug, our technique scans the training data and finds a small set of training items such that if these items were altered the bug would be either solved or at least mitigated. We call this set of data the *fix* and in addition to helping fix the selected test errors it should also be small enough to be interpretable. These two criteria thus define a notion of optimality for the fix.

There has been some prior work in the general area of simplifying the iterative model building cycle. The work by Kulesza et al. (2010) provided explanations for certain types of errors, and several works have visually represented different model errors as in Brooks et al. (2015). Systems such as Amershi et al. (2014) and Kulesza et al. (2012) are designed to bridge the gap between detecting errors and making improvements . Many of these works achieved impressive results but were specifically tailored towards a specific type of human comprehensible data sets (music, natural language documents) or models that are sufficiently simple to be understood by a user. Our work build on these by creating a generic, principled framework.

2. Definitions

Let $\mathcal{M}_D$ be a model trained on the training data $D = (X, Y)$. The model $\mathcal{M}_D$ is said to have a bug if it makes wrong prediction on some test examples. We denote a subset of test examples that are bugs as $D^* = (X^*, Y^*)$, where Y^* are the correct labels. We assume that the cause of the bug lies in the training data itself, not in e.g. the choice of machine learning model. A *fix* to a bug set D^* is a change to the training data D such that if the model is trained on

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

the modified dataset, it will be more accurate when applied to D^*. For simplicity, we restrict the changes to just operating on the labels of training examples. As such, a fix is defined by the change vector Δ on training labels. We denote by $D \oplus \Delta$ the modified dataset obtained by applying the fix Δ to the dataset D; the operator $\oplus$ can be e.g. addition for regression and XOR for binary classification.

A fix Δ should satisfy two conditions: (i) it should fix the problem, that is, $\mathcal{M}_{D\oplus\Delta}$ should perform better on the set D^* compared to the original model $\mathcal{M}_D$; (ii) the fix should be small in the sense that Δ is sparse. This has two reasons: First, as a debugging aid, it should provide a concise representation that will aid developers in the debugging process. Moreover, assuming the learning algorithm is stable (Bousquet & Elisseeff, 2002), a "small" fix will ensure that $\mathcal{M}_{D\oplus\Delta}$ will maintain the accuracy of $\mathcal{M}_D$ while correcting the predictions on D^*.

Therefore, we introduce two functions. $\mathit{Complexity}\,(\Delta)$ measures the complexity or the size of the fix. For example, in binary classification Δ indicates which labels should be flipped so the complexity can be the number of non-zero elements; in regression it can be the norm of the change vector. $\mathit{Fixloss}\,(D, D^*, \Delta)$ measures the loss of the model $\mathcal{M}_{D\oplus\Delta}$ on D^*. A better training set $D \oplus \Delta$ will incur lower Fixloss. We will often shorthand the function as $\mathit{Fixloss}\,(\Delta)$ in the interest of clarity. Both functions are problem dependent.

Our main optimization problem to find the optimal fix is:

$$\operatorname*{arg\,min}_{\Delta\in\mathbf{\Delta}} \left[\mathit{Fixloss}\,(\Delta) + \lambda \cdot \mathit{Complexity}\,(\Delta)\right] \quad (1)$$

This formulation is completely general and applicable to any supervised learners, but is usually a combinatorial optimization problem. We note its similarity to the machine teaching problem (Zhu, 2015).

3. Analytical Solutions for Certain Learners

In this section we show an efficient way to find the optimal fix Δ for both Ordinary Least Square (OLS) regression and Gaussian Processes (GP) regression.

3.1. OLS Regression

In the case of OLS, the instances are vectors in $\mathtt{R}^d$ and the labels are scalars in $\mathtt{R}$. Therefore, the training set D can be thought of as a matrix X of size $n \times d$ of the instances and a label vector Y of length n. Denoting by β_D the weight vector of the regression model trained on D, we recall that (Duda et al., 2012) $\beta_D = (X^TX)^{-1}X^TY$. Now, let us consider the result we would get if we retrained with a fix. Consider a fix Δ and corresponding altered data set $D \oplus \Delta$, then training a model using $D \oplus \Delta$ instead of D we get

$$\beta_{D\oplus\Delta} = (X^TX)^{-1}X^T(Y+\Delta) = \beta_D + A\Delta, \quad (2)$$

where $A = (X^TX)^{-1}X^T$.

The natural formulation of fixloss for this situation should be the loss function used by the underlying OLS (sum of squares): hence we use the squared L2 norm on the test set error.

$$\mathit{Fixloss}\,(D, D^*, \Delta) = \left\| Y^* - {X^*}^T\beta_{D\oplus\Delta} \right\|_2^2$$
$$= \left\| Y^* - \left({X^*}^T\beta_D + {X^*}^T A\Delta \right) \right\|_2^2$$

We let the complexity function be the p-norm of Δ, usually $p = 1$ or 2: $\mathit{Complexity}\,(\Delta) = \|\Delta\|_p$. Substituting these back into (1) we find the optimal fix by

$$\operatorname*{arg\,min}_{\Delta\in\mathbf{\Delta}} \left\| Y^* - \left({X^*}^T\beta_D + {X^*}^T A\Delta \right) \right\|_2^2 + \lambda \|\Delta\|_p .$$

This can be further simplified by defining $\hat{Y} = Y^* - {X^*}^T\beta_D$ and $\hat{A} = {X^*}^T A$ to obtain

$$\operatorname*{arg\,min}_{\Delta\in\mathbf{\Delta}} \left\| \hat{Y} - \hat{A}\Delta \right\|_2^2 + \lambda \|\Delta\|_p \quad . \quad (3)$$

Therefore, finding the optimal Δ reduces to solving a regularized linear regression problem. This is a convex optimization problem for $p \geq 1$.

The case of $p = 1$ is of special interest since it promotes sparse fixes. If D^* contains a single bug then $\hat{Y}$ is a scalar and $\hat{A}$ is a n-dimensional vector where n is the number of training items. In this case, (3) boils down to $\operatorname*{arg\,min}_{\Delta} \left\| \hat{y} - \hat{A}\Delta \right\|_2^2 + \lambda \|\Delta\|_1$ and it is easy to verify that a fix Δ is optimal iff $\Delta_i = 0$ for every i such that $\left|\hat{A}_i\right| < \max_j \left|\hat{A}_j\right|$. Assuming that all the values of $\left|\hat{A}_j\right|$ are unique, we get a natural ranking of proposed fixes by the order of the value of $\left|\hat{A}_j\right|$. Recalling that

$$\hat{A} = X^*A = X^*(X^TX)^{-1}X^T$$

we note that $\hat{A}$ is the Mahalanobis inner product (Kung, 2014) between the test data and training data. Hence we note the interesting relation that the closer some training point Z is to X^* under the similarity metric induced by this Mahalanobis inner product, the more favorable it is to fix the label of Z in order to fix the prediction of X^*. We return to the concept of a similarity metric in Section 4, in which we extend the discussion to more complicated learning techniques such as neural nets and ensembles of trees.

3.2. Gaussian Processes

In this section we show that the techniques presented in Section 3.1 can be extended to the setting of Gaussian Processes (Rasmussen & Williams, 2006). Namely, given a training set D and a set of bugs D^*, a kernel function K and an assumed Gaussian noise on the training labels with variance σ^2 we once more ask to optimize (1) .

Letting $\hat{Y}$ be the MAP predictor for the Gaussian Process on the test points, its formulation as given by equation 2.23 in (Rasmussen & Williams, 2006) is below:

$$\hat{Y} = K(X^*, X)(K(X, X) + \sigma I)^{-1}Y \tag{4}$$

Therefore, denoting $\hat{A} = K(X^*, X)(K(X, X) + \sigma I)^{-1}$ we obtain that $\hat{Y} = \hat{A}Y$. Now we want to add the label delta Δ to Y to minimize the evaluation equation. Note that $\hat{A}$ does not depend on the value of Y and so remains unchanged. Hence, the fixloss function is $\mathit{Fixloss}\,(\Delta) = \left\|Y^* - \hat{Y}\right\| = \left\|(Y^* - \hat{A}Y) - \hat{A}\Delta\right\|$ which when substituted into (1) gives the optimization problem

$$\min_{\Delta \in \mathbf{\Delta}} \left\| \left(Y^* - \hat{A}Y\right) - \hat{A}\Delta \right\| + \lambda \left\|\Delta\right\|$$

Once more minimizing the evaluation function has reduced to a linear regression problem. Considering $\hat{A}$ as a training set, $\left(Y^* - \hat{A}Y\right)$ as the target labels and Δ as the vector representation of the linear regressor shows us this equivalence. Hence, obtaining the optimal fix for a Gaussian Process model is very similar to the case for OLS.

4. Empirical results on advanced learners

In the previous section we were able to present optimal fixes for some learning algorithms since they provided a closed form solution for computing the model. However many commonly used algorithms, such as back propagation for learning neural networks, boosting and random forests do not share this property. In the absence of any other options this would imply that proper optimization of the fixloss function of any fix would have to be achieved by naively applying the delta to the training set, retraining, evaluating the fixloss it produced and doing this over the entire space of permissible values of Δ. Even if there is a way to limit the search space for Δ, evaluating its fixloss for a learning algorithm like a deep neural net would be prohibitively expensive since it would require a total retraining of the model. In this section we provide a framework for working around this problem and provide empirical results for an example scenario where the learner is a regression tree ensemble.

In these situations one solution is to simplify by introducing the idea of a *heuristic model similarity* (HMS). A heuristic model similarity is defined for a specific learning algorithm $\mathcal{A}$ and evaluates the similarity between a training point x and test point x^*, drawn from train and test sets X and X^* respectively, from the perspective of a model $\mathcal{M}$. We will denote the similarity between x_i and x_j^* as s_{ij}, where $s_{ij}$is defined as (assuming that the algorithm and model are clear from context):

$$s_{ij} = HMS_{\mathcal{A}}\left(\mathcal{M}, x_i, x_j^*\right)$$
$$s_i = \sum_j HMS_{\mathcal{A}}\left(\mathcal{M}, x_i, x_j^*\right)$$

This choice was motivated by the discussion at the end of section 3.1 where ordering the training points by a Mahalanobis inner product with respect to a test point produced a ranked list of training data points with the largest fixloss improvement if their corresponding label was changed. If we assume that the heuristic similarity value s_i for a training data point x_i does approximately correspond to how much it will improve fixloss (as detailed below):

$$\mathit{Fixloss}\,(\Delta) \approx loss(\mathcal{M}_D, D^*) - \sum_i s_i\Delta_i \tag{5}$$

Then our optimization can be restated as

$$\min_{\Delta \in \mathbf{\Delta}} \quad -(\textstyle\sum_i s_i\Delta_i) + \lambda(\mathit{Complexity}\,(\Delta)) \tag{6}$$

This considerably simplifies finding Δ since we can evaluate each data point individually instead of potentially having to evaluate every possible subset of training points at the cost of obtaining only approximately optimal results.

Given the lack of an analytical solution, understanding which heuristic is appropriate for a given learner becomes an important consideration. We design an experimental methodology for this called the *ease of change analysis*. Given an initial training set D on which we train $\mathcal{M}_D$, define D^* as the set of test points on which $\mathcal{M}_D$ returns the wrong prediction. We can now compare a set of heuristics $\{HMS_i\}$ by performing a set of trials in which we randomly select $d^* \in D^*$, compute the optimal Δ_i for each heuristic on this point with the constraint $\mathit{Complexity}\,(\Delta_i) < c_j$ for some complexity budget c_j and inspect the average $\mathit{Fixloss}\,(\Delta_i)$ over these trials. This simulates the process of understanding a bug, with heuristics returning higher fixloss scores providing a set of training points that better explain the error. A superior HMS should obtain superior fixloss scores on a wide range of complexity budgets c_j. We illustrate this in an empirical example using a regression tree ensemble.

4.1. Evaluating heuristics on a regression tree ensemble

In this experiment we demonstrate the suitability of three different heuristic model similarities for debugging the regression tree ensemble learner. The first HMS was simply the Euclidean metric on the feature representation of two points. The second HMS (Jaccard) assigned scores based on how similar a path two data points traverse from the root of a tree to a leaf. The third HMS (Weighted-Jaccard) builds on the Jaccard method by including information about the internal nodes of each regression tree. In order to evaluate these three HMSs we use a publicly available flight delay dataset.[1]. This dataset contains information about flights such as their time, date, destination airport, carrier as the features and a boolean label indicating whether the flight experienced over 15 minutes of delay.

The dataset is unbalanced with around 20% being positive. We selected a training set consisting of 500,000 examples and a test set of 100,000 examples. We then trained two initial tree ensembles (one with 20 trees and one with 100 trees) on this set and obtained an initial D^* set for both. We use gradient boosting (MART) as the learning algorithm. The complexity budget (defined as the cardinality of Δ to encourage sparsity and interpretability) constraints were set as 10, 25 ,50, 75 and 100: we determined any larger set would be hard for a user to inspect. Finally we took several hundred random trials for each complexity budget and compared the fixloss of the different Δ_i vectors. Given the unbalanced nature of the dataset we aggregated results for false negative d^* separately from false positives.

Figure 1 shows that for both the false negative bugs and false positive bugs that the two HMS implementations that account for model structure unsurprisingly beat the model agnostic Euclidean implementation. For a given budget of $Complexity\,(\Delta)$ we see that the average fixloss achieved on the y-axis by the two Jaccard metrics is often several times higher. Moreover, we also noted that retraining on a given Δ has no significant effect on the model's aggregate performance on the rest of the test set. This methodology shows how different HMS implementations can be tested on different dataset for a given learning algorithm to ensure that the returned Δ vectors are a useful tool for debugging this family of models.

5. Discussion

In this work we suggested a framework for helping machine learning developers understand "bugs" in a learned model by returning pertinent training items. This framework has closed-form solutions for certain learners and for more complex learning algorithms we presented a heuristic solution. The framework can be applied to any type of supervised learner and is designed to scale gracefully with respect to the size of both model and data. We hope that this will make it a useful framework in practical industrial situations.

[1]http://datamarket.azure.com/dataset/oakleaf/us_air_carrier_flight_delays_incr

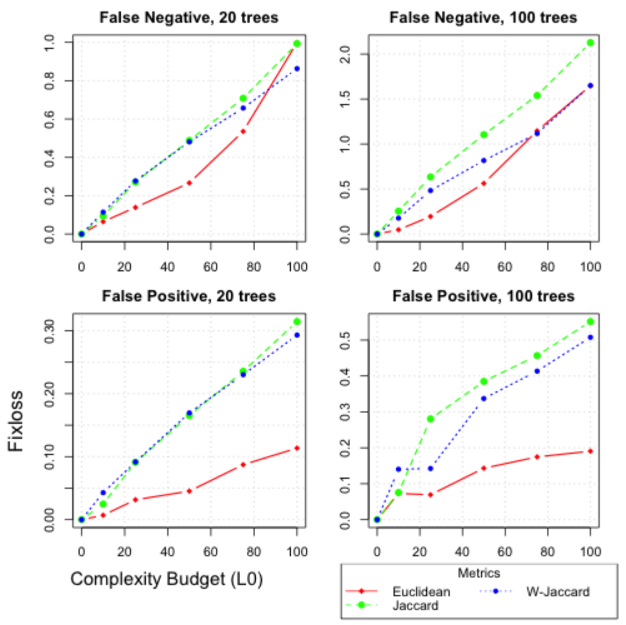

Figure 1. Ease of loss analysis for different test conditions, where a higher value for any given x-value indicating greater fixloss performance. Both Jaccard HMS variants comprehensively outperform a model agnostic version at lower complexity budgets and reach at least parity at higher budgets.

There are many interesting avenues to continue exploring. For example, it is sometimes useful to be able to provide an explanation to the prediction a model makes. We could imagine this in the case of doctor disagreeing with a diagnostic model and wanting to understand what prompted this diagnosis before making a final decision. Our methodology supports providing such explanations as a set of similar examples from the training data. Another investigation might focus on performing a human study to quantify the relationship between the quality of fix data and its impact on human debugging performance. Many other debugging methods are possible and it is our hope that this study will encourage others to further explore tools to assist the machine learning model development cycle.

References

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

Bousquet, Olivier and Elisseeff, André. Stability and generalization. *The Journal of Machine Learning Research*, 2:499–526, 2002.

Brooks, Michael, Amershi, Saleema, Lee, Bongshin, Drucker, Steven M, Kapoor, Ashish, and Simard, Patrice. Featureinsight: Visual support for error-driven feature ideation in text classification. In *IEEE Symposium on Visual Analytics Science and Technology (VAST)*, 2015.

Duda, Richard O, Hart, Peter E, and Stork, David G. *Pattern classification*. John Wiley & Sons, 2012.

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

Kulesza, Todd, Stumpf, Simone, Burnett, Margaret, and Kwan, Irwin. Tell me more?: the effects of mental model soundness on personalizing an intelligent agent. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1–10. ACM, 2012.

Kung, Sun Yuan. *Kernel methods and machine learning*. Cambridge University Press, 2014.

Rasmussen, Carl Edward and Williams, Christopher K. I. *Gaussian processes for machine learning*. MIT press, 2006.

Zhu, X. Machine teaching: an inverse problem to machine learning and an approach toward optimal education. *AAAI*, 2015.