Summary on A few Testing Machine learning papers

Presented by : Ji Gao

1Department of Computer Science, University of Virginia
https://qdata.github.io/deep2Read/

August 26, 2018
Overview

1. The challenge of verification and testing of machine learning
2. Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints
3. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
4. Understanding Black-box Predictions via Influence Functions
5. An Empirical Study of Bugs in Machine Learning Systems
6. What’s your ML Test Score? A rubric for ML production systems
8. A Family of Test Adequacy Criteria for Database-Driven Applications
9. On testing non-testable programs
## Table: Summarized papers

<table>
<thead>
<tr>
<th>Name</th>
<th>Author</th>
<th>Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepXplore: Automated Whitebox Testing of Deep Learning Systems</td>
<td>Kexin Pei, Yinzhi Cao, Junfeng Yang, Suman Jana (Columbia)</td>
<td>Arxiv</td>
</tr>
<tr>
<td>(Section [??]) Properties of Machine Learning Applications for Use in Metamorphic Testing.</td>
<td>Christian Murphy, Gail Kaiser, Lifeng Hu, Leon Wu (Columbia)</td>
<td>SEKE, 2008</td>
</tr>
<tr>
<td>(Section [5]) An Empirical Study of Bugs in Machine Learning Systems</td>
<td>Ferdian Thung, Shaowei Wang, David Lo, Lingxiao Jiang (Singapore Management University)</td>
<td>2012 International Symposium on Software Reliability Engineering</td>
</tr>
</tbody>
</table>
1. The challenge of verification and testing of machine learning

2. Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints

3. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

4. Understanding Black-box Predictions via Influence Functions

5. An Empirical Study of Bugs in Machine Learning Systems

6. What’s your ML Test Score? A rubric for ML production systems


8. A Family of Test Adequacy Criteria for Database-Driven Applications

9. On testing non-testable programs
The challenge of verification and testing of machine learning

- **Testing**: Evaluating the system in several conditions and observing its behavior, watching for defects.
- **Verification**: Producing a compelling argument that the system will not misbehave under a very broad range of circumstances.
The challenge of verification and testing of machine learning

- Machine learning practitioners have traditionally relied primarily on testing: Accuracy.
- No guarantee for upper bound.
Adversarial samples: Slightly perturbed samples that lead to an error

- Become a problem
- Testing can’t solve this problem
- Even testing on adversarial samples can’t reveal the robustness of the model
- Need stronger guarantee
Several recent works:


Doesn’t seem very practical to me:

Use SAT solver.

Only works on very small networks.

Need to have specific activation function and limit to certain structure.
Outline

1. The challenge of verification and testing of machine learning
2. Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints
3. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
4. Understanding Black-box Predictions via Influence Functions
5. An Empirical Study of Bugs in Machine Learning Systems
6. What’s your ML Test Score? A rubric for ML production systems
8. A Family of Test Adequacy Criteria for Database-Driven Applications
9. On testing non-testable programs
Abstract: Language is increasingly being used to define rich visual recognition problems with supporting image collections sourced from the web. Structured prediction models are used in these tasks to take advantage of correlations between co-occurring labels and visual input but risk inadvertently encoding social biases found in web corpora. In this work, we study data and models associated with multilabel object classification and visual semantic role labeling. We find that (a) datasets for these tasks contain significant gender bias and (b) models trained on these datasets further amplify existing bias. For example, the activity cooking is over 33% more likely to involve females than males in a training set, and a trained model further amplifies the disparity to 68% at test time. We propose to inject corpus-level constraints for calibrating existing structured prediction models and design an algorithm based on Lagrangian relaxation for collective inference. Our method results in almost no performance loss for the underlying recognition task but decreases the magnitude of bias amplification by 47.5% and 40.5% for multilabel classification and visual semantic role labeling, respectively.
Visual recognition tasks: Mining relationships in the image

However, bias in the data makes learner learns stereotypes.

Cooking in dataset: 33% man and 66% woman

Cooking in a unlabeled data by Learned annotator: 16% man and 84% woman

Stereotypes got amplified
Identifying stereotypes

Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints

- Bias score:
  \[ b(\text{verb}, \text{man}) = \frac{c(\text{verb}, \text{man})}{c(\text{verb}, \text{man}) + c(\text{verb}, \text{woman})} \]

- Bias Amplification:
  \[ \frac{1}{|O|} \sum_{o} (\hat{b}(o, g) - b^*(o, g)) \]

\( \hat{b} \) is the bias of learned annotator, and \( b^* \) is the bias of original data. \( o \) is the output, and \( g \) is gender.
Debiasing the model

Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints

- We can force the bias in a certain range, but it’s hard.
- Form this as an optimization problem.
- Can be optimized together with the original optimization process.
Debiasing the model

Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints

- Find \( \arg \max_{y \in Y} f_\theta(y, i), f_\theta(y, i) \) is a scoring function, in this case is \( \log p(y|i, \theta) \)

- Constraints \( b^* - \gamma \leq \frac{\sum_i y_{v=v^*, r \in M}^i}{\sum_i y_{v=v^*, r \in M}^i + \sum_i y_{v=v^*, r \in W}^i} \leq b^* + \gamma \), i is the index of test sample

- Use combinatorial optimization tools to solve.
Experiment
Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints
## Experiment result

### Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints

<table>
<thead>
<tr>
<th>Method</th>
<th>Viol.</th>
<th>Amp. bias</th>
<th>Perf. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vSRL: Development Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRF</td>
<td>154</td>
<td>0.050</td>
<td>24.07</td>
</tr>
<tr>
<td>CRF + RBA</td>
<td>107</td>
<td>0.024</td>
<td>23.97</td>
</tr>
<tr>
<td>vSRL: Test Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRF</td>
<td>149</td>
<td>0.042</td>
<td>24.14</td>
</tr>
<tr>
<td>CRF + RBA</td>
<td>102</td>
<td>0.025</td>
<td>24.01</td>
</tr>
<tr>
<td>MLC: Development Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRF</td>
<td>40</td>
<td>0.032</td>
<td>45.27</td>
</tr>
<tr>
<td>CRF + RBA</td>
<td>24</td>
<td>0.022</td>
<td>45.19</td>
</tr>
<tr>
<td>MLC: Test Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRF</td>
<td>38</td>
<td>0.040</td>
<td>45.40</td>
</tr>
<tr>
<td>CRF + RBA</td>
<td>16</td>
<td>0.021</td>
<td>45.38</td>
</tr>
</tbody>
</table>
Discussion

Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints

- Original method works properly, but still amplifies the stereotypes
- Possible to build a test set to exploit such bias?
1. The challenge of verification and testing of machine learning
2. Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints
3. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
4. Understanding Black-box Predictions via Influence Functions
5. An Empirical Study of Bugs in Machine Learning Systems
6. What’s your ML Test Score? A rubric for ML production systems
8. A Family of Test Adequacy Criteria for Database-Driven Applications
9. On testing non-testable programs
Abstract
Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Abstract: The blind application of machine learning runs the risk of amplifying biases present in data. Such a danger is facing us with word embedding, a popular framework to represent text data as vectors which has been used in many machine learning and natural language processing tasks. We show that even word embeddings trained on Google News articles exhibit female/male gender stereotypes to a disturbing extent. This raises concerns because their widespread use, as we describe, often tends to amplify these biases. Geometrically, gender bias is first shown to be captured by a direction in the word embedding. Second, gender neutral words are shown to be linearly separable from gender definition words in the word embedding. Using these properties, we provide a methodology for modifying an embedding to remove gender stereotypes, such as the association between between the words receptionist and female, while maintaining desired associations such as between the words queen and female. We define metrics to quantify both direct and indirect gender biases in embeddings, and develop algorithms to "debias" the embedding. Using crowd-worker evaluation as well as standard benchmarks, we empirically demonstrate that our algorithms significantly reduce gender bias in embeddings while preserving the its useful properties such as the ability to cluster related concepts and to solve analogy tasks. The resulting embeddings can be used in applications without amplifying gender bias.
Introduction

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

- Word Embedding trained include sexism result: 
  \[ \text{man} \rightarrow \text{woman} \approx \text{computer programmer} \rightarrow \text{homemaker} \]
- As widely used, word embedding can even amplify the gender stereotypes.
- Example: For a search algorithm use word embedding, if male names have a closer distance to the word "programmer" than female names, it would likely to return a male name first.
Goal of this paper

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Major goals:
- Identify gender stereotypes.
- Debiasing.
Gender bias in language has been studied over decades in various fields.

Several work on “fair” binary classification, including multiple ways to modify the prediction algorithm.

Previous work dealing with gender bias by completely remove gender from the dataset.
Identify gender stereotypes in word embedding
Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

- Step 1: Find out pair of words that are similar to the pair (she, he).
  Similar metric: Cosine similarity

$$ S_{(a,b)}(x, y) = \cos(\vec{a} - \vec{b}, \vec{x} - \vec{y}) \text{ if } ||\vec{x} - \vec{y}|| \leq \delta $$

**Extreme she occupations**

1. homemaker
2. nurse
3. receptionist
4. librarian
5. socialite
6. hairdresser
7. nanny
8. bookkeeper
9. stylist
10. housekeeper
11. interior designer
12. guidance counselor

**Extreme he occupations**

1. maestro
2. skipper
3. protege
4. philosopher
5. captain
6. architect
7. financier
8. warrior
9. broadcaster
10. magician
11. fighter pilot
12. boss
Identify gender stereotypes in word embedding
Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Step 2: Use crowdsourcing to identify whether pair of words are stereotypes or not.

- **Gender stereotype she-he analogies.**
  - sewing-carpentry
  - nurse-surgeon
  - blond-burly
  - giggle-chuckle
  - sassy-snappy
  - volleyball-football
  - register-nurse-physician
  - interior designer-architect
  - feminism-conservatism
  - vocalist-guitarist
  - diva-superstar
  - cupcakes-pizzas
  - housewife-shopkeeper
  - softball-baseball
  - cosmetics-pharmaceuticals
  - petite-lanky
  - charming-affable
  - hairdresser-barber

- **Gender appropriate she-he analogies.**
  - queen-king
  - waitress-waiter
  - sister-brother
  - ovarian cancer-prostate cancer
  - mother-father
  - convent-monastery
Identify gender direction
Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

- Is there a universal gender vector exists?
- Aggregate different pair of words that reflect gender, i.e., he-she, mother-father, woman-man.
- Use PCA to capture the major direction.
If a word should be gender-neutral, then the bias can be measured by

$$\text{DirectBias}_c = \frac{1}{|N|} \sum_{\omega \in N} |\cos(\omega, g)|^c$$

g is the unit gender vector, and $c$ as a hyperparameter.
What if two words which should have gender relationship to each other? The bias can be measured by projecting the embedding vector onto the gender space. Suppose $w = w_g + w_\perp$

$$\beta(w, v) = \left(w \cdot v - \frac{w_\perp \cdot v_\perp}{\|w_\perp\|_2 \|v_\perp\|_2}\right)/w \cdot v$$
Debiasing

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

- Step 1: Identify the gender subspace
- Step 2: Define two methods: Hard de-biasing and Soft de-biasing.
  - Hard debiasing: Neutralize all words to the gender space. Equalize the distance of pair of words outside the space.
  - Soft debiasing: Tradeoff the previous method with similarity to the original embedding
Debiasing (Contd.)

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

**Hard Debiasing:**

**Step 2a: Hard de-biasing (neutralize and equalize).** Additional inputs: words to neutralize $N \subseteq W$, family of equality sets $\mathcal{E} = \{E_1, E_2, \ldots, E_m\}$ where each $E_i \subseteq W$. For each word $w \in N$, let $\bar{w}$ be re-embedded to

$$\bar{w} := (\bar{w} - \bar{w}_B)/\|\bar{w} - \bar{w}_B\|.$$

For each set $E \in \mathcal{E}$, let

$$\mu := \sum_{w \in E} w/|E|$$

$$\nu := \mu - \mu_B$$

For each $w \in E$, $\bar{w} := \nu + \sqrt{1-\|
u\|^2} \frac{\bar{w}_B - \mu_B}{\|\bar{w}_B - \mu_B\|}$
Debiasing Result

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Effectiveness: Is stereotypes successfully removed after the debiasing?
Debiasing Result
Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Accuracy: Will debiasing affect the performance of the word embedding?

<table>
<thead>
<tr>
<th></th>
<th>RG</th>
<th>WS</th>
<th>analogy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before</td>
<td>62.3</td>
<td>54.5</td>
<td>57.0</td>
</tr>
<tr>
<td>Hard-debiased</td>
<td>62.4</td>
<td>54.1</td>
<td>57.0</td>
</tr>
<tr>
<td>Soft-debiased</td>
<td>62.4</td>
<td>54.2</td>
<td>56.8</td>
</tr>
</tbody>
</table>

According to the result, the answer is no.
Outline

1. The challenge of verification and testing of machine learning
2. Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints
3. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
4. Understanding Black-box Predictions via Influence Functions
5. An Empirical Study of Bugs in Machine Learning Systems
6. What’s your ML Test Score? A rubric for ML production systems
8. A Family of Test Adequacy Criteria for Database-Driven Applications
9. On testing non-testable programs
Abstract: How can we explain the predictions of a blackbox model? In this paper, we use influence functions, a classic technique from robust statistics, to trace a model’s prediction through the learning algorithm and back to its training data, thereby identifying training points most responsible for a given prediction. To scale up influence functions to modern machine learning settings, we develop a simple, efficient implementation that requires only oracle access to gradients and Hessian-vector products. We show that even on non-convex and non-differentiable models where the theory breaks down, approximations to influence functions can still provide valuable information. On linear models and convolutional neural networks, we demonstrate that influence functions are useful for multiple purposes: understanding model behavior, debugging models, detecting dataset errors, and even creating visually indistinguishable training-set attacks.
How can we explain the predictions on machine learning models?

Intuition: Understand what the consequence will be if change training samples

Idea: Borrow the **Influence functions** idea from robust statistics
Machine learning: Learn from $X \rightarrow Y$

Suppose $L(z, \theta)$ is the loss function, $\theta$ stands for parameters, $z$ stands for training point $(x, y)$

total loss $\frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta)$, $z_i...z_n$ are samples

$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta)$

Goal: Figure out the change of $\theta$ and $L(\theta)$ if a single point $z$ changed
Influence Function
Understanding Black-box Predictions via Influence Functions

- influence function: if one sample is upweighted, how would the prediction result change?
- That is \( \hat{\theta}' = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i; \theta) + \epsilon L(z; \theta) \)
- The result from paper (Cook and Weisberg, 1982) is:
- \( \mathcal{I}_{up,\text{params}}(z) := \frac{d\hat{\theta}}{d\epsilon}|_{\epsilon=0} = -H_{\hat{\theta}}^{-1} \nabla_{\theta} L(z, \hat{\theta}) \), Here \( H_{\hat{\theta}} \) is the Hessian matrix: \( H_{\hat{\theta}} = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta}^2 L(z_i; \hat{\theta}). \)
Using influence Function to estimate the effect of removing a sample

Understanding Black-box Predictions via Influence Functions

- If $\hat{\theta}' = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z, \theta)$, we have
  $$\frac{d\hat{\theta}}{d\epsilon}|_{\epsilon=0} = -H^{-1}_{\hat{\theta}} \nabla_{\theta} L(z, \hat{\theta})$$
- Removing a sample is equivalent to $\epsilon = -\frac{1}{n}$
- When $n$ is large, $\frac{1}{n} \approx 0$
- Therefore, $\hat{\theta}' - \hat{\theta} \approx -\frac{1}{n} \frac{d\hat{\theta}}{d\epsilon}|_{\epsilon=0} = \frac{1}{n} H^{-1}_{\hat{\theta}} \nabla_{\theta} L(z, \hat{\theta})$
- Similarly, for any test sample $z_{test}$, the change of $L(z_{test}, \theta)$ can be approximated by:
  $$L(z_{test}, \theta)' - L(z_{test}, \theta) \approx -\frac{1}{n} \frac{dL(z_{test}, \theta)}{d\epsilon}|_{\epsilon=0}$$
  $$= -\frac{1}{n} \nabla_{\theta} L(z_{test}, \theta) \frac{d\hat{\theta}}{d\epsilon}|_{\epsilon=0}$$
  $$= \frac{1}{n} \nabla_{\theta} L(z_{test}, \theta) H^{-1}_{\hat{\theta}} \nabla_{\theta} L(z, \hat{\theta})$$
For perturbation on a data point $z \rightarrow z_\delta$, here $x_\delta = x + \delta$

Can be viewed as adding a new training point while deleting the old point.

\[
\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z_\delta, \theta) - \epsilon L(z, \theta)
\]

We have

\[
\mathcal{I}_{pert,\text{params}}(z) = \mathcal{I}_{up,\text{params}}(z_\delta) - \mathcal{I}_{up,\text{params}}(z)
\]

\[
= - H^{-1}_\hat{\theta} \nabla_\theta (L(z_\delta, \hat{\theta}) - L(z, \hat{\theta}))
\]

Assume it’s continuous, then the perturbation can be approximated as

\[
\hat{\theta}_{z_\delta, -z} - \hat{\theta} \approx - H^{-1}_\hat{\theta} [\nabla_x \nabla_\theta L(z_\delta, \hat{\theta})] \delta
\]
Let $p(y|x) = \sigma(y \theta^T x)$. For a training point $z = (x, y)$,
$L(z, \theta) = \log(1 + \exp(-y \theta^T x))$

What’s the influence on a test point?
$I_{up,loss}(z, z_{test}) = -y_{test} y \cdot \sigma(-y_{test} \theta^T x_{test}) \cdot (-y \theta^T x) \cdot x_{test}^T H_{\hat{\theta}}^{-1} x$

$-y \theta^T x$ stands for error

$H_{\hat{\theta}}^{-1}$ decides the impact, the size of the impact on the gradient direction.

$x \cdot x_{test}$ represents the distance
Calculation the hessian matrix takes $O(np^2 + p^3)$ time, and we also want to calculate it over all training points, which takes extra n times.

Idea: Use Hessian-vector products (HVP) to approximate the hessian. That is, instead of $H_\hat{\theta}^{-1}$, calculate the product $H_\hat{\theta}^{-1} \nabla_\theta L(z, \hat{\theta})$ directly.

Transform the matrix inversion to an optimization problem, and use conjugate gradient to solve it.

In addition, do a stochastic estimation on the Hessian matrix.
Compare the result of influence function and retraining, matches.
Previous result assume we can achieve the optimized solution. For Neural Network, it’s not convex, so the method doesn’t work?

Use a convex approximation:
\[
\tilde{L}(z, \theta) = L(z, \tilde{\theta}) + \nabla L(z, \tilde{\theta})^T (\theta - \tilde{\theta}) + \frac{1}{2} (\theta - \tilde{\theta})^T (H_{\tilde{\theta}} + \lambda I)(\theta - \tilde{\theta})
\]
How to handle Non-differentialable loss?

Idea: generate a smooth approximation.

For example, Hingeloss  \( Hinge(s) = \max(0, 1 - s) \)

approximated with  \( SmoothHinge(s) = t\log(1 + \exp\left(\frac{1-s}{t}\right)) \)

Result:

![Smooth approximations to the hinge loss. (a) By varying \( t \), we can approximate the hinge loss with arbitrary accuracy: the green and blue lines are overlaid on top of each other. (b) Using a random, wrongly-classified test point, we compared the predicted vs. actual differences in loss after leave-one-out retraining on the 100 most influential training points. A similar trend held for other test points. The SVM objective is to minimize 0.005 \( ||w||^2 + \frac{1}{n} \sum_i Hinge(y_i, w^T x_i) \). Left: Influence functions were unable to accurately predict the change, overestimating its magnitude considerably. Mid: Using SmoothHinge(\( t \), 0.001) let us accurately predict the change in the hinge loss after retraining. Right: Correlation remained high over a wide range of \( t \), though it degrades when \( t \) is too large. When \( t = 0.001 \), Pearson’s R = 0.95; when \( t = 0.1 \), Pearson’s R = 0.91.](chart)
Case study 1: Understanding which sample is important to current prediction

If a train sample have a larger influence (positive $-I_{up,loss}$), it’s more helpful to the prediction.

Result show in the graph.
Case study 2: Poisoning sample

For a typical test sample, modify the training sample following the rule of

\[ z'_i := \prod (z'_i + \alpha \text{sign}(I_{pert,loss}(z'_i, z_{test}))) \]

A slight modification on the training samples can change the prediction result!

Figure 5. Training-set attacks. We targeted a set of 30 test images featuring the first author’s dog in a variety of poses and backgrounds. By maximizing the average loss over these 30 images, we created a visually-imperceptible change to the particular training image (shown on top) that flipped predictions on 16 test images.
Case study 3: Debugging domain mismatch

- Domain mismatch: Difference between the training set and testing set.
- Experiment: Predict diabetic patients to be readmitted using logistic regression with 127 features. Originally, 3 out of 24 children under age 10 in training set is readmitted. Manually remove 20 data from the set, so currently it’s 3 out of 4. A bias is created between training and test set, and many children data in the test set is wrongly predicted in the test set.
- Without influence function: Check the learned $\theta$ on the feature children, not very abnormal: 14 features have a larger coefficient.
- Use influence function: All 4 children samples left have a very strong influence on those wrongly predicted samples, 30-40 times larger than others. Therefore there exists a domain mismatch.
**Case study 4: Fixing mislabeled examples**

- **Idea:** mislabeled examples have a negative effect on the prediction.
- **Experiment:** Manually random flip 10% data.
- **Baseline1:** Check data points randomly
- **Baseline2:** Check those data points with highest loss.
- **Method:** Check those data points that would have a positive effect if removed using influence function.
Influence Function: In different statistics papers (Cook, 1977; Cook & Weisberg, 1980; 1982)

Several previous papers in ML use influence functions to study model robustness and do fast cross-validation in kernel methods (Christmann & Steinwart (2004); Debruyne et al. (2008); Liu et al. (2014))

Training set attack is equivalent to previous training set attack (Biggio et al. (2012)), and several poisoning attack papers (Mei & Zhu, 2015; Li et al., 2016).

(Cadamuro et al. (2016)) consider the task of taking an incorrect test prediction and finding a small subset of training data such that changing the labels on this subset makes the prediction correct.
1. The challenge of verification and testing of machine learning
2. Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints
3. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
4. Understanding Black-box Predictions via Influence Functions
5. An Empirical Study of Bugs in Machine Learning Systems
6. What’s your ML Test Score? A rubric for ML production systems
8. A Family of Test Adequacy Criteria for Database-Driven Applications
9. On testing non-testable programs
Abstract: Many machine learning systems that include various data mining, information retrieval, and natural language processing code and libraries are used in real world applications. Search engines, internet advertising systems, product recommendation systems are sample users of these algorithm-intensive code and libraries. Machine learning code and toolkits have also been used in many recent studies on software mining and analytics that aim to automate various software engineering tasks. With the increasing number of important applications of machine learning systems, the reliability of such systems is also becoming increasingly important. A necessary step for ensuring reliability of such systems is to understand the features and characteristics of bugs occurred in the systems. A number of studies have investigated bugs and fixes in various software systems, but none focuses on machine learning systems. Machine learning systems are unique due to their algorithm-intensive nature and applications to potentially large-scale data, and thus deserve a special consideration. In this study, we fill the research gap by performing an empirical study on the bugs in machine learning systems. We analyze three systems, Apache Mahout, Lucene, and OpenNLP, which are data mining, information retrieval, and natural language processing tools respectively. We look into their bug databases and code repositories, analyze a sample set of bugs and corresponding fixes, and label the bugs into various categories. Our study finds that 22.6% of the bugs belong to the algorithm/method category, 15.6% of the bugs belong to the non-functional category, and 13% of the bugs belong to the assignment-initialization category. We also report the relationship between bug categories and bug severities, the time and effort needed to fix the bugs, and bug impacts. We highlight several bug categories that deserve attention in future research.
Goal of this paper

An Empirical Study of Bugs in Machine Learning Systems

- Intuition: Bugs are prevalent in software systems. None of the previous work focus on Machine Learning systems
- Goal
  1. understand the nature of the bugs in machine learning systems
  2. help to deal with bugs
An Empirical Study of Bugs in Machine Learning Systems

Design

Analyze 3 machine learning systems/libraries.

- Apache Mahout: A data mining library
- Apache Lucene: A information retrieval library.
- Apache OpenNLP: A NLP toolkit

Contributions:

- first to analyze on machine learning systems
- manually categorize 500 bug reports
- investigate the relationship between bug categories and bug severities, bug fixing time and effort, and bug impact.
Method - Bug categorization

An Empirical Study of Bugs in Machine Learning Systems

Categories: From paper *Defect categorization: making use of a decade of widely varying historical data*

Categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm/method</td>
<td>The implementation of an algorithm/method does not follow the expected behavior.</td>
</tr>
<tr>
<td>assignment_INITIALIZATION</td>
<td>Error in assigning variable values.</td>
</tr>
<tr>
<td>checking</td>
<td>Missing necessary checks that lead to an error or a wrong error message.</td>
</tr>
<tr>
<td>data</td>
<td>Wrong use of data structure.</td>
</tr>
<tr>
<td>external interface</td>
<td>Error in interfacing with other systems or users, such as using deprecated methods from other systems, required updates to own external interfaces for ease of usage, etc.</td>
</tr>
<tr>
<td>internal interface</td>
<td>Error in interfacing with another component of the same system, such as violating the contract of inheritance, wrong use of operations from other classes, etc.</td>
</tr>
<tr>
<td>logic</td>
<td>Incorrect expressions in conditional statements (e.g., if, while, etc.)</td>
</tr>
<tr>
<td>non-functional</td>
<td>Violations in non-functional requirements, such as improper variable or method names, wrong documentation to the implementation of a method, etc.</td>
</tr>
<tr>
<td>timing/optimization</td>
<td>Error that causes concurrency or performance issues, such as deadlock, high memory usage, etc.</td>
</tr>
<tr>
<td>configuration</td>
<td>Error in non-code (e.g., configuration files) that affects functionality.</td>
</tr>
<tr>
<td>others</td>
<td>Other bugs that do not fall into one of the above categories.</td>
</tr>
</tbody>
</table>
RQ1: Bug frequency

Lucene have higher bug density than Mahout and OpenNLP.

Table III

<table>
<thead>
<tr>
<th>Application</th>
<th>Bug Count Per kLOC</th>
<th>Bug Count Per Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mahout</td>
<td>1.79 bugs/kLOC</td>
<td>73.36 bugs/year</td>
</tr>
<tr>
<td>Lucene</td>
<td>2.77 bugs/kLOC</td>
<td>144.76 bugs/year</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>1.45 bugs/kLOC</td>
<td>95.68 bugs/year</td>
</tr>
</tbody>
</table>
### Table IV

**Bug Types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm/method</td>
<td>113</td>
<td>22.60%</td>
</tr>
<tr>
<td>non-functional</td>
<td>78</td>
<td>15.60%</td>
</tr>
<tr>
<td>assignment_INITIALIZATION</td>
<td>65</td>
<td>13.00%</td>
</tr>
<tr>
<td>checking</td>
<td>57</td>
<td>11.40%</td>
</tr>
<tr>
<td>external interface</td>
<td>38</td>
<td>7.60%</td>
</tr>
<tr>
<td>internal interface</td>
<td>38</td>
<td>7.60%</td>
</tr>
<tr>
<td>data</td>
<td>28</td>
<td>5.60%</td>
</tr>
<tr>
<td>logic</td>
<td>27</td>
<td>5.40%</td>
</tr>
<tr>
<td>configuration</td>
<td>27</td>
<td>5.40%</td>
</tr>
<tr>
<td>timing/OPTIMIZATION</td>
<td>24</td>
<td>4.80%</td>
</tr>
<tr>
<td>others</td>
<td>5</td>
<td>1%</td>
</tr>
</tbody>
</table>

Algorithm > non-functional > assignment
RQ3: Bug severities

<table>
<thead>
<tr>
<th>Type</th>
<th>Severity</th>
<th>Count</th>
<th>Proportion</th>
<th>Type</th>
<th>Severity</th>
<th>Count</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm/method</td>
<td>Blocker</td>
<td>1</td>
<td>0.88%</td>
<td>internal interface</td>
<td>Blocker</td>
<td>1</td>
<td>2.63%</td>
</tr>
<tr>
<td></td>
<td>Critical</td>
<td>3</td>
<td>2.65%</td>
<td></td>
<td>Critical</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Major</td>
<td>70</td>
<td>61.95%</td>
<td></td>
<td>Major</td>
<td>22</td>
<td>57.89%</td>
</tr>
<tr>
<td></td>
<td>Minor</td>
<td>33</td>
<td>29.20%</td>
<td></td>
<td>Minor</td>
<td>11</td>
<td>28.95%</td>
</tr>
<tr>
<td></td>
<td>Trivial</td>
<td>6</td>
<td>5.31%</td>
<td></td>
<td>Trivial</td>
<td>4</td>
<td>10.53%</td>
</tr>
<tr>
<td>assignment_INITIALIZATION</td>
<td>Blocker</td>
<td>2</td>
<td>3.08%</td>
<td>logic</td>
<td>Blocker</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Critical</td>
<td>1</td>
<td>1.54%</td>
<td></td>
<td>Critical</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Major</td>
<td>34</td>
<td>52.31%</td>
<td></td>
<td>Major</td>
<td>15</td>
<td>55.56%</td>
</tr>
<tr>
<td></td>
<td>Minor</td>
<td>24</td>
<td>36.92%</td>
<td></td>
<td>Minor</td>
<td>8</td>
<td>29.63%</td>
</tr>
<tr>
<td></td>
<td>Trivial</td>
<td>4</td>
<td>6.15%</td>
<td></td>
<td>Trivial</td>
<td>4</td>
<td>14.81%</td>
</tr>
<tr>
<td>checking</td>
<td>Blocker</td>
<td>2</td>
<td>3.51%</td>
<td>non-functional</td>
<td>Blocker</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Critical</td>
<td>1</td>
<td>1.75%</td>
<td></td>
<td>Critical</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Major</td>
<td>32</td>
<td>56.14%</td>
<td></td>
<td>Major</td>
<td>38</td>
<td>48.72%</td>
</tr>
<tr>
<td></td>
<td>Minor</td>
<td>17</td>
<td>29.82%</td>
<td></td>
<td>Minor</td>
<td>29</td>
<td>37.18%</td>
</tr>
<tr>
<td></td>
<td>Trivial</td>
<td>5</td>
<td>8.77%</td>
<td></td>
<td>Trivial</td>
<td>11</td>
<td>14.10%</td>
</tr>
<tr>
<td>configuration</td>
<td>Blocker</td>
<td>0</td>
<td>0.00%</td>
<td>timing/optimization</td>
<td>Blocker</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Critical</td>
<td>0</td>
<td>0.00%</td>
<td></td>
<td>Critical</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Major</td>
<td>19</td>
<td>70.37%</td>
<td></td>
<td>Major</td>
<td>19</td>
<td>79.17%</td>
</tr>
<tr>
<td></td>
<td>Minor</td>
<td>6</td>
<td>22.22%</td>
<td></td>
<td>Minor</td>
<td>5</td>
<td>20.83%</td>
</tr>
<tr>
<td></td>
<td>Trivial</td>
<td>2</td>
<td>7.41%</td>
<td></td>
<td>Trivial</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>data</td>
<td>Blocker</td>
<td>0</td>
<td>0.00%</td>
<td>others</td>
<td>Blocker</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Critical</td>
<td>1</td>
<td>3.57%</td>
<td></td>
<td>Critical</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Major</td>
<td>15</td>
<td>53.57%</td>
<td></td>
<td>Major</td>
<td>3</td>
<td>60.00%</td>
</tr>
<tr>
<td></td>
<td>Minor</td>
<td>12</td>
<td>42.86%</td>
<td></td>
<td>Minor</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Trivial</td>
<td>0</td>
<td>0.00%</td>
<td></td>
<td>Trivial</td>
<td>2</td>
<td>40.00%</td>
</tr>
<tr>
<td>external interface</td>
<td>Blocker</td>
<td>1</td>
<td>2.63%</td>
<td></td>
<td>Blocker</td>
<td>1</td>
<td>2.63%</td>
</tr>
<tr>
<td></td>
<td>Critical</td>
<td>1</td>
<td>2.63%</td>
<td></td>
<td>Critical</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>Major</td>
<td>21</td>
<td>55.26%</td>
<td></td>
<td>Major</td>
<td>21</td>
<td>55.26%</td>
</tr>
<tr>
<td></td>
<td>Minor</td>
<td>12</td>
<td>31.58%</td>
<td></td>
<td>Minor</td>
<td>12</td>
<td>31.58%</td>
</tr>
<tr>
<td></td>
<td>Trivial</td>
<td>3</td>
<td>7.89%</td>
<td></td>
<td>Trivial</td>
<td>3</td>
<td>7.89%</td>
</tr>
</tbody>
</table>
Table VI
**Bug-Fixing Durations In Terms of Days**

<table>
<thead>
<tr>
<th>Type</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm/method</td>
<td>0.0022</td>
<td>2433.7033</td>
<td>91.7238</td>
<td>3.8740</td>
</tr>
<tr>
<td>assignment-initialization</td>
<td>0.0003</td>
<td>160.9271</td>
<td>9.9160</td>
<td>0.5000</td>
</tr>
<tr>
<td>checking</td>
<td>0.0017</td>
<td>195.7766</td>
<td>17.1335</td>
<td>1.1175</td>
</tr>
<tr>
<td>configuration</td>
<td>0.0016</td>
<td>195.9011</td>
<td>22.3583</td>
<td>2.8032</td>
</tr>
<tr>
<td>data</td>
<td>0.0014</td>
<td>676.3825</td>
<td>40.8279</td>
<td>2.2666</td>
</tr>
<tr>
<td>external interface</td>
<td>0.0006</td>
<td>1700.5871</td>
<td>69.2463</td>
<td>0.4275</td>
</tr>
<tr>
<td>internal interface</td>
<td>0.0029</td>
<td>1688.5275</td>
<td>93.3543</td>
<td>2.4852</td>
</tr>
<tr>
<td>logic</td>
<td>0.0016</td>
<td>59.8305</td>
<td>6.8892</td>
<td>1.2537</td>
</tr>
<tr>
<td>non-functional</td>
<td>0.0006</td>
<td>1330.4142</td>
<td>47.8057</td>
<td>0.6949</td>
</tr>
<tr>
<td>timing/optimization</td>
<td>0.0017</td>
<td>569.7309</td>
<td>71.6649</td>
<td>3.3596</td>
</tr>
<tr>
<td>others</td>
<td>0.0005</td>
<td>1.3128</td>
<td>0.3344</td>
<td>0.0594</td>
</tr>
</tbody>
</table>
### Table X
**Bug-Fixing Effort (Detailed)**

<table>
<thead>
<tr>
<th>Type</th>
<th>Effort</th>
<th>Count</th>
<th>Proportion</th>
<th>Type</th>
<th>Effort</th>
<th>Count</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm/method</td>
<td>1</td>
<td>71</td>
<td>62.83%</td>
<td>internal interface</td>
<td>1</td>
<td>25</td>
<td>65.79%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>17</td>
<td>15.04%</td>
<td></td>
<td>2</td>
<td>3</td>
<td>7.89%</td>
</tr>
<tr>
<td></td>
<td>&gt;2</td>
<td>25</td>
<td>22.12%</td>
<td></td>
<td>&gt;2</td>
<td>10</td>
<td>26.32%</td>
</tr>
<tr>
<td>assignment_INITIALIZATION</td>
<td>1</td>
<td>51</td>
<td>78.46%</td>
<td>logic</td>
<td>1</td>
<td>22</td>
<td>81.84%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11</td>
<td>16.92%</td>
<td></td>
<td>2</td>
<td>4</td>
<td>14.81%</td>
</tr>
<tr>
<td></td>
<td>&gt;2</td>
<td>3</td>
<td>4.62%</td>
<td></td>
<td>&gt;2</td>
<td>1</td>
<td>3.70%</td>
</tr>
<tr>
<td>checking</td>
<td>1</td>
<td>32</td>
<td>56.14%</td>
<td>non-functional</td>
<td>1</td>
<td>54</td>
<td>69.23%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16</td>
<td>28.07%</td>
<td></td>
<td>2</td>
<td>14</td>
<td>17.95%</td>
</tr>
<tr>
<td></td>
<td>&gt;2</td>
<td>9</td>
<td>15.79%</td>
<td></td>
<td>&gt;2</td>
<td>10</td>
<td>12.82%</td>
</tr>
<tr>
<td>configuration</td>
<td>1</td>
<td>16</td>
<td>59.26%</td>
<td>timing/optimization</td>
<td>1</td>
<td>6</td>
<td>25.00%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td>25.93%</td>
<td></td>
<td>2</td>
<td>10</td>
<td>41.67%</td>
</tr>
<tr>
<td></td>
<td>&gt;2</td>
<td>4</td>
<td>14.81%</td>
<td></td>
<td>&gt;2</td>
<td>8</td>
<td>33.33%</td>
</tr>
<tr>
<td>data</td>
<td>1</td>
<td>16</td>
<td>57.14%</td>
<td>others</td>
<td>1</td>
<td>5</td>
<td>100.00%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>8</td>
<td>28.57%</td>
<td></td>
<td>2</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>&gt;2</td>
<td>4</td>
<td>14.29%</td>
<td></td>
<td>&gt;2</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>external interface</td>
<td>1</td>
<td>26</td>
<td>68.42%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>13.16%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt;2</td>
<td>7</td>
<td>18.42%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table XI

**Numbers of Impacted Files Per Bug**

<table>
<thead>
<tr>
<th>Type</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm/method</td>
<td>1</td>
<td>117</td>
<td>11.7434</td>
<td>3</td>
</tr>
<tr>
<td>assignment_INITIALIZATION</td>
<td>1</td>
<td>74</td>
<td>5.2459</td>
<td>2</td>
</tr>
<tr>
<td>checking</td>
<td>1</td>
<td>115</td>
<td>7.2982</td>
<td>2</td>
</tr>
<tr>
<td>configuration</td>
<td>1</td>
<td>40</td>
<td>5.2692</td>
<td>1</td>
</tr>
<tr>
<td>data</td>
<td>1</td>
<td>124</td>
<td>12.3571</td>
<td>3</td>
</tr>
<tr>
<td>external interface</td>
<td>1</td>
<td>92</td>
<td>13.3514</td>
<td>3</td>
</tr>
<tr>
<td>internal interface</td>
<td>1</td>
<td>52</td>
<td>7.6842</td>
<td>4.5</td>
</tr>
<tr>
<td>logic</td>
<td>1</td>
<td>15</td>
<td>2.7778</td>
<td>2</td>
</tr>
<tr>
<td>non-functional</td>
<td>1</td>
<td>676</td>
<td>16.7143</td>
<td>2</td>
</tr>
<tr>
<td>others</td>
<td>1</td>
<td>47</td>
<td>10.2000</td>
<td>1</td>
</tr>
<tr>
<td>timing/optimization</td>
<td>1</td>
<td>69</td>
<td>11.5833</td>
<td>5</td>
</tr>
</tbody>
</table>
Abstract: Using machine learning in real-world production systems is complicated by a host of issues not found in small toy examples or even large offline research experiments. Testing and monitoring are key considerations for assessing the production-readiness of an ML system. But how much testing and monitoring is enough? We present an ML Test Score rubric based on a set of actionable tests to help quantify these issues.
- Real world machine learning systems might get problem not found in small **toy examples** or even large **offline experiments**.
- This paper list a set of actionable tests, and design a scoring system to measure how ready for production a given machine learning system is.
Scoring

For each test, one point is awarded for executing the test manually and documenting and distributing the results. A second point is awarded if there is a system in place to run that test automatically on a repeated basis. The final ML Test Score is computed by taking the minimum of the scores aggregated for every test.

- 0 points: More of a research project than a productionized system.
- 1-2 points: Not totally untested, but it is worth considering the possibility of serious holes in reliability.
- 3-4 points: There's been first pass at basic productionization, but additional investment may be needed.
- 5-6 points: Reasonably tested, but it's possible that more of those tests and procedures may be automated.
- 7-10 points: Strong levels of automated testing and monitoring, appropriate for mission-critical systems.
- 12+ points: Exceptional levels of automated testing and monitoring.
Related Work

Machine learning systems differ from traditional software-based systems in that the behavior of ML systems is not specified directly in code but is learned from data.

- **Distribution**: Test that the distributions of each feature match your expectations.
- **Correlation**: Test the relationship between each feature and the target, and the pairwise correlations between individual signals.
- **Calculation cost**: Test the cost of each feature.
- **Validity**: Test that a model does not contain any features that have been manually determined as unsuitable for use.
- **Privacy**: Test that your system maintains privacy controls across its entire data pipeline.
- **Develop time**: Test the calendar time needed to develop and add a new feature to the production model.
- **Code**: Test all code that creates input features, both in training and serving.
Tests for Model Development

It can be all too tempting to rely on a single-number summary metric to judge performance, perhaps masking subtle areas of unreliability.

- **Code:** Test that every model specification undergoes a code review and is checked in to a repository.
- **Metric:** Test the relationship between offline proxy metrics and the actual impact metrics.
- **Hyperparameters:** Test the impact of each tunable hyperparameter.
- **Staleness:** Test the effect of model staleness.
- **Test against a simpler model as a baseline.**
- **Test model quality on important data slices.**
- **Test the model for implicit bias.**
Tests for ML Infrastructure

- Test the reproducibility of training. Use the same data to train twice.
- Unit test model specification code.
- Integration test the full ML pipeline.
- Test model quality before attempting to serve it.
- Test that a single example or training batch can be sent to the model, and changes to internal state can be observed from training through to prediction.
- Test models via a canary process before they enter production serving environments.
- Test how quickly and safely a model can be rolled back to a previous serving version.
Monitoring Tests for ML

- Test for upstream instability in features, both in training and serving.
- Test that data invariants hold in training and serving inputs.
- Test that your training and serving features compute the same values.
- Test for model staleness.
- Test for NaNs or infinities appearing in your model during training or serving.
- Test for dramatic or slow-leak regressions in training speed, serving latency, throughput, or RAM usage.
- Test for regressions in prediction quality on served data.
1. The challenge of verification and testing of machine learning
2. Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints
3. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
4. Understanding Black-box Predictions via Influence Functions
5. An Empirical Study of Bugs in Machine Learning Systems
6. What’s your ML Test Score? A rubric for ML production systems
8. A Family of Test Adequacy Criteria for Database-Driven Applications
9. On testing non-testable programs

Abstract: ... We design, implement, and evaluate DeepXplore, the first white-box framework for systematically testing real-world DL systems. We address two main problems: (1) generating inputs that trigger different parts of a DL systems logic and (2) identifying incorrect behaviors of DL systems without manual effort. First, we introduce neuron coverage for systematically estimating the parts of DL system exercised by a set of test inputs. Next, we leverage multiple DL systems with similar functionality as cross-referencing oracles and thus avoid manual checking for erroneous behaviors. We demonstrate how finding inputs triggering differential behaviors while achieving high neuron coverage for DL algorithms can be represented as a joint optimization problem and solved efficiently using gradient-based optimization techniques ...
Goal of this paper

- It is important to ensure Deep Learning system works well.
- Problem: Traditional testing methods of Deep Learning have problems: Expensive labeling and Low neuron coverage coverage
- Automatic test case generation with high neuron coverage
Goal of this paper

Figure 1: An example erroneous behavior of Nvidia DAVE-2 self-driving car platform found by DeepXplore. The DNN-based self-driving car correctly decides to turn left for image (a); by contrast, the car decides to turn right and crashes into the guardrail for image (b), a slightly darker version of (a).

Figure 3: Comparison of program flows of a traditional program and a neural network. The nodes in gray denote the corresponding basic blocks or neurons that participated while processing an input.
Maximizing neuron coverage: For each step, select an inactive neuron, and try to increase its value.

Labeling: Use differential testing. Train multiple target DNN together and find those test cases that are supported by all DNNs except one.
Experiment Design


Metrics:

- Neuron coverage
- Execution time
- Retraining accuracy

Not very appealing as the major experiment result is on neuron coverage: Some metric defined by this paper.
Next step on this paper

- Use test case generated to improve the neural network:
  (In the paper) retraining
  Select out nodes that are not effective, remove/change them.

- The two things optimized together doesn’t seems very compatible. It is possible that only optimize one thing will achieve better result.
1. The challenge of verification and testing of machine learning
2. Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints
3. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
4. Understanding Black-box Predictions via Influence Functions
5. An Empirical Study of Bugs in Machine Learning Systems
6. What's your ML Test Score? A rubric for ML production systems
8. A Family of Test Adequacy Criteria for Database-Driven Applications
9. On testing non-testable programs
A Family of Test Adequacy Criteria for Database-Driven Applications

Abstract: Although a software application always executes within a particular environment, current testing methods have largely ignored these environmental factors. Many applications execute in an environment that contains a database. In this paper, we propose a family of test adequacy criteria that can be used to assess the quality of test suites for database-driven applications. Our test adequacy criteria use dataflow information that is associated with the entities in a relational database. Furthermore, we develop a unique representation of a database-driven application that facilitates the enumeration of database interaction associations. These associations can reflect an application's definition and use of database entities at multiple levels of granularity. The usage of a tool to calculate intraprocedural database interaction associations for two case study applications indicates that our adequacy criteria can be computed with an acceptable time and space overhead.
Many software applications include databases.

Current testing methods ignore specific environment.

Goal of this paper is to design test accuracy criteria on database-driven approaches.
Test cases should reveal faults of database applications.

Specifically, this paper uses database integrity to represent the faults:

**Validity:**
1. It inserts a record into a database that does not reflect the real world.
2. It fails to insert a record into the database when the status of the real world changes.

**Completeness:**
1. It deletes a record from a database when this record still reflects the real world.
2. The status of the real world changes and it fails to include this information as a record inside of a database.
Method
A Family of Test Adequacy Criteria for Database-Driven Applications

- Traditional def-use test doesn’t consider interaction with database.
- For the database case, should be doing an enumeration:

\[ \text{Definition } 8. \text{ A test suite } T \text{ for database interaction control flow graph } G_{DB} = (N_{DB}, E_{DB}) \text{ satisfies the all-database-DUs test adequacy criterion if and only if for each association } \langle n_d, n_{use}, x \rangle, \text{ where } x \in D(I) \text{ and } n_d, n_{use} \in N_{DB}, \text{ there exists a test in } T \text{ to create a complete path } \pi_x \text{ in } G_{DB} \text{ that covers the association.} \Box \]

- Also an algorithm to generate the new CFG graph including database interactions.
Future directions:

- Use this method as a guideline of automatic input generation.
- Perform static checking on such applications.
1. The challenge of verification and testing of machine learning
2. Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints
3. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
4. Understanding Black-box Predictions via Influence Functions
5. An Empirical Study of Bugs in Machine Learning Systems
6. What’s your ML Test Score? A rubric for ML production systems
8. A Family of Test Adequacy Criteria for Database-Driven Applications
9. On testing non-testable programs
Abstract: A frequently invoked assumption in program testing is that there is an oracle (i.e. the tester or an external mechanism can accurately decide whether or not the output produced by a program is correct). A program is non-testable if either an oracle does not exist or the tester must expend some extraordinary amount of time to determine whether or not the output is correct. The reasonableness of the oracle assumption is examined and the conclusion is reached that in many cases this is not a realistic assumption. The consequences of assuming the availability of an oracle are examined and alternatives investigated.
Oracle problem
On testing non-testable programs

- Non-testable programs:
  1. There’s no oracle.
  2. The oracle is practically too difficult to find.
- Is it able to test without the oracle?
Testing without an oracle
On testing non-testable programs

- Produce a pseudo-oracle - dual coding
- Reduce the test set.
- Accept the plausible result