Improved Influence Function for Error Detection

Thang Nguyen-Duc[*], **Hoang Thanh-Tung**[*]
FPT Software AI Center
`{thangnd34, tunght18}@fsoft.com.vn`

Quan Tran
Adobe Research
`quan.tran@adobe.com`

Anh Dau, Nghi Bui
FPT Software AI Center
`{anhdtv7, nghibdq}@fsoft.com.vn`

Abstract

Influence function (IF) is a powerful tool for estimating the influence of a data point on another data point. IF is widely used in machine learning, especially for detecting anomalous examples. However, empirical results suggest that IF is unstable when it is applied to deep networks. In this paper, we provide an explanation of the instability of IF and propose a solution to this problem. We show that IF is particularly unreliable when the two data points belong to two different classes or their representations are far from each other. Based on this analysis, we propose a modification to the way IF is used in detection tasks. Experiments show that our modification significantly improves the performance and stability of IF on downstream tasks while incurring no additional computational cost.

1 Introduction

Influence function (IF) (Koh and Liang, 2017) is an instance-based approach to understanding black-box predictors, especially deep neural networks. Different from the feature-based approach, which identifies parts of the input that are responsible for the model's prediction, instance-based approach identifies which inputs are the most influential to the model's prediction. When its assumptions are (approximately) satisfied, IF can give a reasonable estimate of the influence of a data point on any other data point. This capability allows IF to detect harmful data points - data points that have a negative influence on other data points. IF is a powerful tool for detecting adversarial (Cohen et al., 2020), poisonous (Cinà et al., 2021), and erroneous (Anonymous, 2022; Dau et al., 2022) example.[1] The success of IF has inspired a series of follow up works on efficient influence estimation (Charpiat et al., 2019; Khanna et al., 2019; Barshan et al., 2020; Pruthi et al., 2020). For brevity, we call IF and its variants influence functions (IFs).

However, IFs are very unstable when applied to deep networks (Basu et al., 2021; Anonymous, 2022). In this paper, we provide a deeper analysis with both empirical and theoretical explanation of the phenomenon. In Sec. 3, we empirically and theoretically show that IF is very noisy when the two data points belong to two different classes, but it becomes much more reliable when the two data points are in the same class.

Motivated by our analysis, we propose a simple method for improving IFs. Compared to the previous error detection algorithms, our algorithm introduces zero computational overhead while being much more stable and performant (Sec. 4). Our improved IFs can be used in many more general settings, not limited to error detection.

Our contributions include

1. An explanation of the instability of IFs. Our analysis sheds light on the behavior of IFs and helps the development of better IFs.

2. A stable, high performance error detection algorithm based on IFs.

2 Background and Related work

We define the notations used in this paper. Let $\mathbf{z} = (\mathbf{x}, \mathbf{y})$ be a data point, where $\mathbf{x} \in \mathcal{X}$ is the input, $\mathbf{y} \in \mathcal{Y}$ is the target output; $\mathcal{Z} = \{\mathbf{z}^{(i)}\}_{i=1}^{n}$ be a dataset of n data points; $\mathcal{Z}_{-j} = \mathcal{Z} \backslash \mathbf{z}^{(j)}$ be the dataset $\mathcal{Z}$ with $\mathbf{z}^{(j)}$ removed; $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ be a model with parameter $\boldsymbol{\theta}$; $\mathcal{L}_{\mathcal{Z},\boldsymbol{\theta}} = \frac{1}{n}\sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{z}^{(i)}; \boldsymbol{\theta})$ be the loss of $f_{\boldsymbol{\theta}}$ on $\mathcal{Z}$, where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^{+}$ is the loss function; $\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{Z},\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\theta}}_{-i} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{Z}_{-i},\boldsymbol{\theta}}$ be the optimal parameters of the model $f_{\boldsymbol{\theta}}$ trained on $\mathcal{Z}$ and $\mathcal{Z}_{-i}$. In this paper, $f_{\boldsymbol{\theta}}$ is a deep networks and $\hat{\boldsymbol{\theta}}$ is found by training f with gradient descent.

[*]Equal contribution

[1]A copy of (Anonymous, 2022) is attached to Appendix E

2.1 Influence function and variants

The influence of a data point $\mathbf{z}^{(i)}$ on another data point $\mathbf{z}^{(j)}$ is defined as

$$s^{(ij)} = \ell(\mathbf{z}^{(j)}; \hat{\boldsymbol{\theta}}_{-i}) - \ell(\mathbf{z}^{(j)}; \hat{\boldsymbol{\theta}}) \quad (1)$$

$s^{(ij)} < 0$ means that removing $\mathbf{z}^{(i)}$ decreases the loss at $\mathbf{z}^{(j)}$. In other words, $s^{(ij)} < 0$ means that $\mathbf{z}^{(i)}$ is harmful to $\mathbf{z}^{(j)}$. Naive computation of $s^{(ij)}$ requires retraining $f_{\boldsymbol{\theta}}$ on $\mathcal{Z}_{-i}$. Averaging this influence score over the dataset $\mathcal{Z}$ gives the influence of $\mathbf{z}^{(i)}$ on the model $f_{\boldsymbol{\theta}}$ trained on $\mathcal{Z}$

$$s^{(i)} = \frac{1}{n}\sum_{j=1}^{n} s^{(ij)}$$

Koh and Liang (2017) used first and second order derivatives to quickly estimate $s^{(ij)}$ without retraining the model $f_{\boldsymbol{\theta}}$ on $\mathcal{Z}_{-i}$

$$\begin{aligned} s^{(ij)} &\approx IF(\mathbf{z}^{(i)}, \mathbf{z}^{(j)}) \\ &\approx \frac{1}{n}\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)}; \hat{\boldsymbol{\theta}})^\top H_{\hat{\boldsymbol{\theta}}}^{-1} \nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(j)}; \hat{\boldsymbol{\theta}}) \end{aligned} \quad (2)$$

Exact computation of $H_{\hat{\boldsymbol{\theta}}}^{-1}$ is intractable for modern networks. Koh and Liang (2017) developed a fast algorithm for estimating $H_{\hat{\boldsymbol{\theta}}}^{-1}\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(j)}; \hat{\boldsymbol{\theta}})$ and used only the derivatives w.r.t. the last layer's parameters to improve the algorithm's speed. Charpiat et al. (2019) proposed the gradient dot product (GD) and gradient cosine similarity (GC) as faster alternatives to IF. Pruthi et al. (2020) argued that the influence can be better approximated by accumulating it through out the training process (TracIn). The formula for IFs are summarized in Tab. 1 in Appendix A.

IFs are measures of the similarity between the gradients of two data points. The intuition of IFs is that gradients of harmful examples are dissimilar from that of normal examples.

2.2 Influence functions for error detection

Influence functions are a powerful tool for detecting anomalous data points. Although in this paper, we focus on the error detection problem, our discussion and algorithm apply to other detection problems as well.

In the error detection problem, we are given a noisy dataset $\mathcal{Z}$ and have to identify erroneous data points in $\mathcal{Z}$. A data point is an error if it has a negative influence on other data points. Alg. 1 uses IFs to measure the influence of a data point $\mathbf{z} \in \mathcal{Z}$ on a set of clean data points $\mathcal{Z}'$. If $\mathbf{z}$ has a negative influence on $\mathcal{Z}'$ then it is likely an error. Alg. 1 ranks the data points in $\mathcal{Z}$ by how harmful they are. The most harmful data points are removed from the dataset or are re-labeled by human. Models trained on the cleaned dataset are expected to have better performance than model trained on the original dataset.

Algorithm 1 Influence function based error detection. Algorithm reproduced with permission from Anonymous (2022).

```
Require:
1: $\mathcal{Z} = \{\mathbf{z}^{(i)}\}_{i=1}^{n}$: a big noisy dataset
2: $\mathcal{Z}' = \{\mathbf{z}'^{(j)}\}_{j=1}^{m}$: a small clean dataset
3: $f_{\hat{\boldsymbol{\theta}}}$: a deep model pretrained on $\mathcal{Z}$
4: $\mathrm{sim}(\cdot,\cdot)$: a similarity measure in Tab. 1
Ensure: $\hat{\mathcal{Z}}$: data points in $\mathcal{Z}$ ranked by score
5: for $\mathbf{z}^{(i)} \in \mathcal{Z}$ do
6:     $s^{(i)} = \frac{1}{m}\sum_{j=1}^{m} \mathrm{sim}(\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)}), \nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}'^{(j)}))$
7: end for
8: $\hat{\mathcal{Z}} = \mathrm{sort}(\mathcal{Z}, \mathrm{key} = \boldsymbol{s}, \mathrm{ascending} = \mathrm{True})$
9: return $\hat{\mathcal{Z}}$
```

3 Improved influence function for error detection

3.1 Motivation

Basu et al. (2021) and Anonymous (2022) empirically observed that influence functions are unstable when they are applied to deep networks. Anonymous (2022) explained this instability as the result of the noise in the feature representation produced by the network. In this section, we provide a deeper analysis of the behavior of influence functions and motivate our solution.

Pezeshkpour et al. (2021); Hanawa et al. (2021) empirically showed that IFs with last layer's gradient perform as well as or better than IFs with all layers' gradient and variants of IF behave similarly. Therefore, for simplicity, we analyze the behavior of GD with last layer's gradient and generalize our results to other IFs. We begin by examining the gradient pattern of an MLP on a 3-class classification problem (Fig. 1). We observe that gradients of wrongly labeled examples are opposite to that of correctly labeled examples of the same original class. A mislabeled data point is likely to have a very negative influence on nearby data points. However, gradients of erroneous data points are not necessarily opposite to that of correct data points

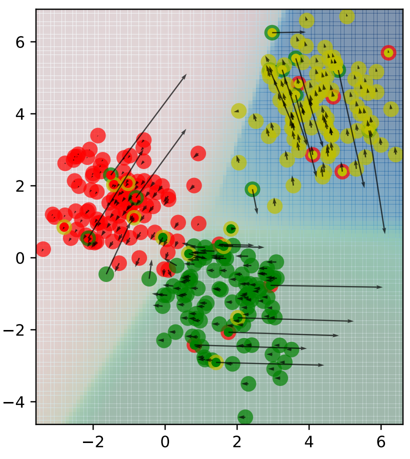


Figure 1: Gradient pattern on a classification problem. The gradient is computed w.r.t. the last layer's parameters. The dataset is generated from 3 Gaussians. We randomly select 20% of the data points and change their labels to random classes. A mislabeled data point is shown by a circle with two colors, the inner color is the original class, the outer color is the new class. We plot only the first 2 dimensions of the gradient. See Appendix C for implementation details and gradient patterns on other dimensions.

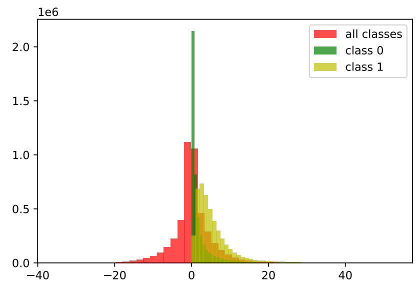


Figure 2: GD score distribution on the IMDB dataset. Results on other datasets are shown in Appendix C.

from other classes. Furthermore, the gradients of two data points from two different classes are almost perpendicular. The effect of a data point from one class on a data point from another class can be very noisy.

We verify the observation on real-world datasets. We compute GD scores of pairs of clean data points from 2 different classes and plot the histogram of the scores. We repeat the same procedure for pairs of data points from each class. Fig. 2 shows the result on IMDB dataset (Maas et al., 2011). We see that in the 2-class case, GD scores are almost normally distributed with a very sharp peak at 0. That means, in many cases, a clean data point from one class has almost no significant effect on data points from the other class. And when it has a significant effect, the effect could be positive or negative with equal probability. In contrast, GD scores of pairs of data points from the same class are almost always positive. A clean data point almost certainly has a positive influence on data points from the same class.

We provide a theoretical explanation of this observation in Appendix D. We formally show that when two clean data points have different labels, their gradients are likely to be perpendicular and the sign of the gradient inner product depends on two noisy quantities. When two clean data points have the same label, the gradient inner product often has a much larger magnitude and its sign depends on only one noisy quantity.

Algorithm 2 Class based influence function for error detection.

```
Require:
 1: $\mathcal{Z} = \{\mathbf{z}^{(i)}\}_{i=1}^{n}$: a big noisy dataset
 2: $C$: number of classes
 3: $\mathcal{Z}'_k = \{\mathbf{z}'^{(j_k)}\}_{j_k=1}^{m_k}$: clean data from class $k$
 4: $\mathcal{Z}' = \bigcup_{k=1}^{C} \mathcal{Z}'_k$: a small clean dataset
 5: $f_{\hat{\boldsymbol{\theta}}}$: a deep model pretrained on $\mathcal{Z}$
 6: $\mathrm{sim}(\cdot,\cdot)$: a similarity measure in Tab. 1
Ensure: $\hat{\mathcal{Z}}$: data points in $\mathcal{Z}$ ranked by score
 7: for $\mathbf{z}^{(i)} \in \mathcal{Z}$ do
 8:     for $k = 1, ..., C$ do
 9:         $s_k^{(i)} = \frac{1}{m_k}\sum_{j=1}^{m_k} \mathrm{sim}(\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)}), \nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}'^{(j_k)}))$
10:     end for
11:     $s^{(i)} = \min_k(s_k^{(i)})$
12: end for
13: $\hat{\mathcal{Z}} = \mathrm{sort}(\mathcal{Z}, \mathrm{key} = \boldsymbol{s}, \mathrm{ascending} = \mathrm{True})$
14: return $\hat{\mathcal{Z}}$
```

3.2 Method

Let's reconsider Alg. 1. The score $s^{(i)}$ in line 6 Alg. 1 can be noisy if the majority of the clean data points $\mathbf{z}'^{(j)}$ do not have the same label as the true label of $\mathbf{z}^{(i)}$. We can make the score less noisy by using clean data points from the same class as the true class of $\mathbf{z}^{(i)}$ only. Because we do not know the true label of $\mathbf{z}^{(i)}$, we compute the influence score of $\mathbf{z}^{(i)}$ on every class in $\mathcal{Z}'$ and select the minimum of these as the influence score of $\mathbf{z}^{(i)}$. We use the minimum because we need to know $\mathbf{z}^{(i)}$ is the most harmful (or unhelpful) to which class. The detailed algorithm is shown in Alg. 2.

The inner for-loop calculates C influence scores.

It calls to the scoring function sim() exactly $|\mathcal{Z}'| = m$ times. The complexity of the inner for-loop in Alg. 2 is equal to that of line 6 in Alg. 1. Thus, the complexity of Alg. 2 is equal to that of Alg. 1.

4 Experiments

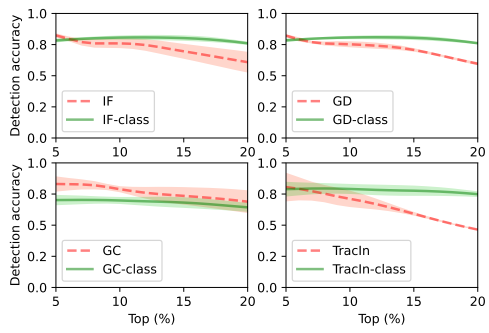


Figure 3: Error detection accuracy on SNLI dataset with $p = 20\%$.

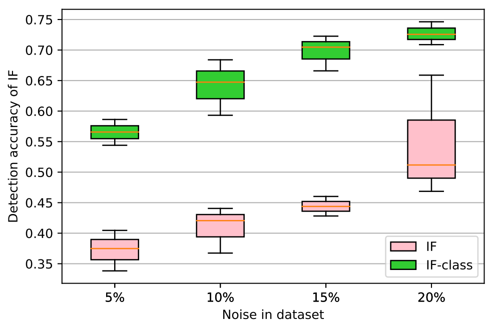


Figure 4: Change in error detection accuracy on the IMDB dataset as the level of noise changes.

We compare our algorithm to the influence function based error detection algorithm from Anonymous (2022) (Alg. 1) on 3 language datasets, namely IMDB (Maas et al., 2011), SNLI (Bowman et al., 2015), and BigCloneBench (Svajlenko et al., 2014). We use BERT (Devlin et al., 2019) for the IMDB and SNLI datasets and CodeBERT (Feng et al., 2020) for the BigClone dataset. Details about the models and datasets can be found in Appendix B. For each dataset, we run our experiments 3 times and report the mean and standard deviation. Results on other datasets is deferred to Appendix C. The patterns reported here are observed in all experiments on all datasets.

We create noisy datasets from the clean datasets above by randomly selecting $p\%$ of the data points and change their labels to random classes. To evaluate an error detection algorithm, we select top $q\%$ most harmful data points from the sorted dataset $\hat{\mathcal{Z}}$ and check how many percent of the selected data points are really erroneous. Intuitively, increasing q allows the algorithm to find more errors but may decrease the detection accuracy.

Fig. 3 shows the error detection accuracy on the SNLI dataset and how the accuracy changes with q. Except for the GC algorithm, our class-based algorithms has higher accuracy and lower variance than the non-class-based versions from (Anonymous, 2022). When q increases, the performance of our algorithms does not decrease as much as that of non-class-based algorithms. This confirms that the class-based IFs are less noisy than the original IFs.

Our class-based influence score fails to improve the performance of GC. Let's reconsider the similarity measure $\text{sim}(\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)}), \nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}'^{(j)}))$. Let's assume that there exist some data points $\mathbf{z}'^{(j)} \in \mathcal{Z}'$ with large gradient $\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}'^{(j)})$. If the similarity measure does not normalize the gradients then $\mathbf{z}'^{(j)}$ will have the dominant effect on the influence score. The noise in the influence score is mostly caused by these data points. GC normalizes the gradient and effectively removes such noise. From Fig. 1, we see that the gradients of erroneous data points tend to be larger than that of normal data points. By normalizing the gradients, GC removes the valuable information about the magnitude of the gradient $\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)})$. That could be harmful to the error detection performance. In Fig. 3, we see that the performance of GC when $p \geq 15\%$ is lower than that of other class-based algorithms.

Fig. 4 shows the change in detection accuracy as the level of noise p goes from 5% to 20%. In this experiment, for each value of p, we set q to be equal to p. Our class-based influence score significantly improves the performance and reduces the variance. We note that when p increases, the error detection problem becomes easier as there are more errors. There detection accuracy, therefore, should increase with p as shown in Fig. 4.

5 Conclusion

In this paper, we study influence functions and identify the source of their instability. We explain our observations using theoretical framework. We introduce a stable, high performance error detection algorithm. Our findings shed light of the development of new influence estimators and on the application of IFs in downstream tasks.

Limitations

Our paper has several limitations

1. We only experimented with the error detection problem. Adversarial and poisonous example detection were not considered in our paper. Testing our class-based algorithm on these problems would significantly improve the quality of our paper. We aim to include these tasks in the extended version of our paper.

2. Our class-based influence score cannot improve the performance of GC algorithm. Although class-based version of GD, IF, and TracIn outperformed the original GC, we aim to develop a stronger version of GC. From the analysis in Sec. 4, we believe that a partially normalized GC could have better performance. In partial GC, we normalize the gradient of the clean data point $\mathbf{z}'^{(j)}$ only. That will remove the noise introduced by $\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}'^{(j)})$ while retaining the valuable information about the norm of $\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)})$.

Ethics Statement

Our paper consider a theoretical aspect of influence functions. It does not have any biases toward any groups of people. Our findings do not cause any harms to any groups of people.

References

Anonymous. 2022. A paper under review. *preprint*.

Elnaz Barshan, Marc-Etienne Brunet, and Gintare Karolina Dziugaite. 2020. Relatif: Identifying explanatory training samples via relative influence. In *International Conference on Artificial Intelligence and Statistics*, pages 1899–1909. PMLR.

Samyadeep Basu, Phil Pope, and Soheil Feizi. 2021. Influence functions in deep learning are fragile. In *International Conference on Learning Representations*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Guillaume Charpiat, Nicolas Girard, Loris Felardos, and Yuliya Tarabalka. 2019. Input similarity from the neural network perspective. *Advances in Neural Information Processing Systems*, 32.

Antonio Emanuele Cinà, Kathrin Grosse, Sebastiano Vascon, Ambra Demontis, Battista Biggio, Fabio Roli, and Marcello Pelillo. 2021. Backdoor learning curves: Explaining backdoor poisoning beyond influence functions. *arXiv preprint arXiv:2106.07214*.

Gilad Cohen, Guillermo Sapiro, and Raja Giryes. 2020. Detecting adversarial samples using influence functions and nearest neighbors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14453–14462.

Anh Dau, Thang Nguyen-Duc, Hoang Thanh-Tung, and Nghi Bui. 2022. Towards using data-centric approach for better code representation learning. *arXiv preprint arXiv:2205.13022*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*.

Kazuaki Hanawa, Sho Yokoi, Satoshi Hara, and Kentaro Inui. 2021. Evaluation of similarity-based explanations. In *International Conference on Learning Representations*.

Rajiv Khanna, Been Kim, Joydeep Ghosh, and Sanmi Koyejo. 2019. Interpreting black box predictions using fisher kernels. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3382–3390. PMLR.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Pouya Pezeshkpour, Sarthak Jain, Byron Wallace, and Sameer Singh. 2021. An empirical comparison of instance attribution methods for NLP. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 967–975, Online. Association for Computational Linguistics.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. 2020. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930.

Jeffrey Svajlenko, Judith F. Islam, Iman Keivanloo, Chanchal K. Roy, and Mohammad Mamun Mia. 2014. Towards a big data curated benchmark of inter-project code clones. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 476–480.

A Additional algorithms and formula

Table 1: Influence function and its variants. We drop the constant factor $1/n$ for clarity.

IF	$\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)};\hat{\boldsymbol{\theta}})^\top H_{\hat{\boldsymbol{\theta}}}^{-1}\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(j)};\hat{\boldsymbol{\theta}})$
GD	$\langle\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)}),\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(j)})\rangle$
GC	$\cos(\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)}),\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(j)}))$
TracIn	$\sum_{t=1}^{T}\eta_t\,\langle\nabla_{\boldsymbol{\theta}^{(t)}}\ell(\mathbf{z}^{(i)}),\nabla_{\boldsymbol{\theta}^{(t)}}\ell(\mathbf{z}^{(j)})\rangle$

B Datasets and models

In this section, we describe the datasets and models in our experiments. We used standard datasets and models and experimented with many different random seeds (namely, 0, 1, and 2). Our results are comparable and reproducible.

B.1 Datasets

IMDB: The dataset includes 50000 reviews from the Internet Movie Database (IMDb) website. The task is a binary sentiment analysis task. The dataset contains an even number of positive and negative reviews. The IMDB dataset is split into training, validation, and test sets of sizes 17500, 7500, and 25000.
SNLI dataset (Standart Natural Language Inference) (Bowman et al., 2015) consists of 570k sentence pairs manually labeled as entailment, contradiction, and neutral. We convert these labels into numbers. It is geared towards serving as a benchmark for evaluating text representational systems.
BigCloneBench (Svajlenko et al., 2014) is a huge code clone benchmark that includes over 6,000,000 true clone pairs and 260,000 false clone pairs from 10 different functionality. The task is to predict whether two pieces of code have the same semantics. This dataset is commonly used in language models for code (Feng et al., 2020; Lu et al., 2021; Guo et al., 2020).

B.2 Models

BERT stands for Bidirectional Encoder Representations from Transformers, is based on Transformers. The BERT model in this paper was pre-trained for natural language processing tasks. We use BERT for IMDB and SNLI datasets.
CodeBert is a bimodal pre-trained model for programming and natural languages. We use CodeBert for BigCloneBench dataset.

C Additional results

C.1 3-class classification experiment

We train a MLP with 2 input neurons, 100 hidden neurons, 3 output neurons with SGD for 1000 epochs. The activation function is LeakyReLU and the learning rate is $\eta = 1e-3$. The last layer has 6 parameters organized into a 3×2 matrix. The gradient of the loss with respect to the last layer's parameters is also organized into a 3×2 matrix. We visualize 3 rows of the gradient matrix in 3 subfigures (Fig. 5).

C.2 Result on IMDB, SNLI, and BigClone

To ensure a fair comparison between our class-based algorithm and algorithm 1, we use the same clean dataset $\mathcal{Z}'$ for both algorithms. The clean dataset $\mathcal{Z}'$ consists of C classes. We have $C = 2$ for the IMDB dataset, $C = 3$ for the SNLI dataset, and $C = 2$ for the BigCloneBench dataset. From each of the C classes, we randomly select $m_k = 50$ $k = 1, ..., C$ clean data points to form $\mathcal{Z}'$. We tried varying m_k from 10 to 1000 and observed no significant changes in performance.

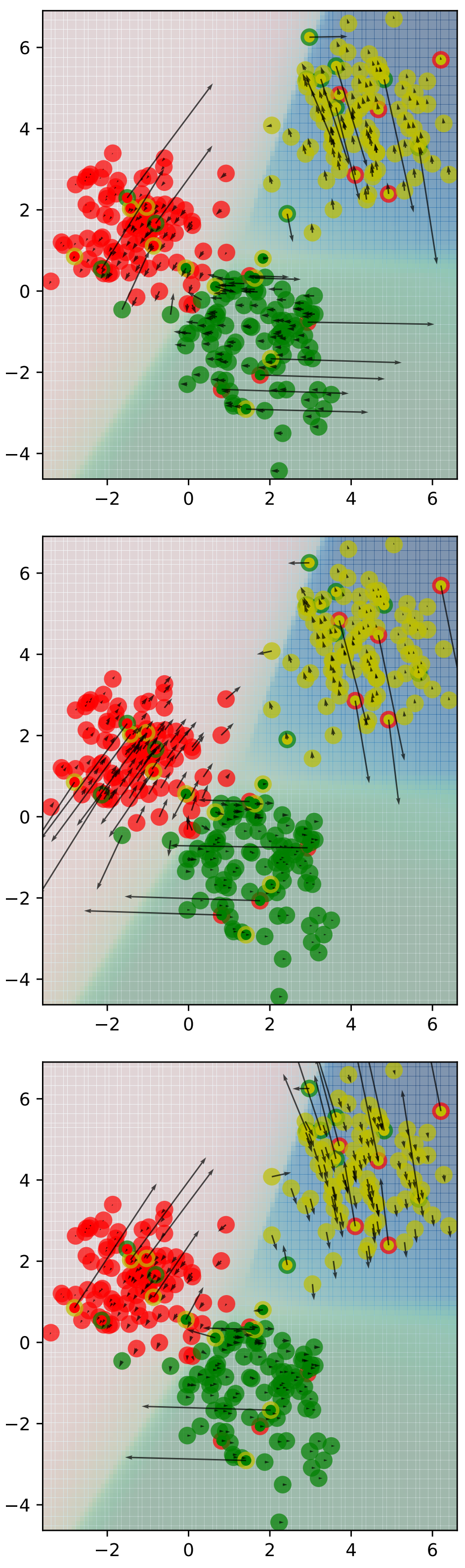


Figure 5: Gradient pattern on a classification problem. Each subfigure shows 2 dimensions of the gradient. The top subfigure shows the 1st and 2nd dimensions of the gradient. The middle subfigure shows the 3rd and 4th dimensions of the gradient. The bottom subfigure shows the 5th and 6th dimensions of the gradient.

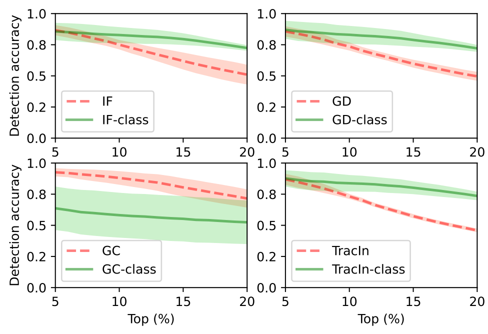


Figure 6: Error detection accuracy on IMDB dataset with $p = 20\%$.

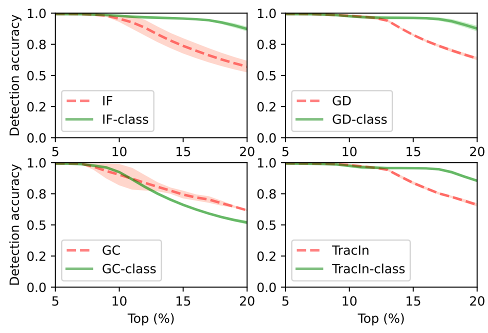


Figure 7: Error detection accuracy on BigCloneBench dataset with $p = 20\%$.

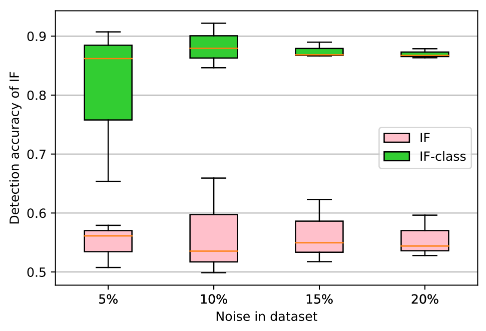


Figure 8: Change in error detection accuracy on the BigCloneBench dataset as the level of noise changes.

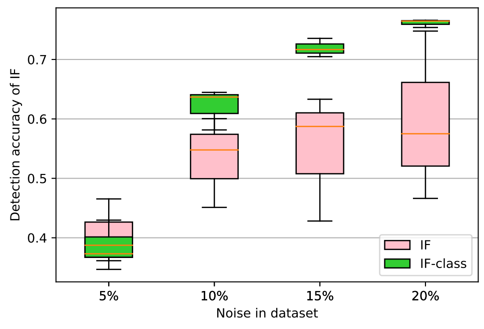


Figure 9: Change in error detection accuracy on the SNLI dataset as the level of noise changes.

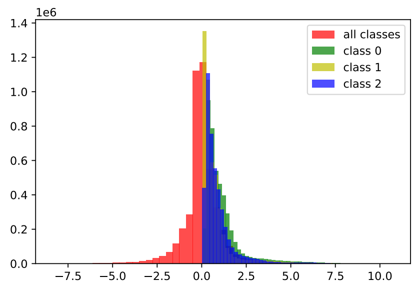


Figure 10: GD score distribution on the SNLI dataset.

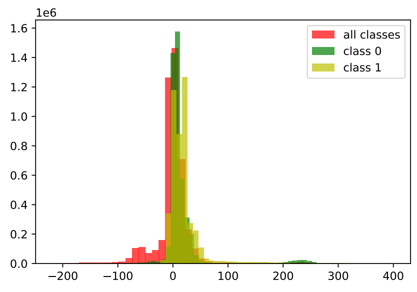


Figure 11: GD score distribution on the BigClone dataset.

D Explanation of the observation in Sec. 3

Let's consider a classification problem with cross entropy loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{d_y} y_i \log \hat{y}_i$$

where d_y is the number of classes. Let $\mathbf{z} = (\mathbf{x}, \mathbf{y})$ be a data point with label k, i.e. $y_k = 1$, $y_i = 0 \, \forall \, i \neq k$. The model $f_{\boldsymbol{\theta}}$ is a deep network with last layer's parameter $W \in \mathbb{R}^{d_y \times d_h}$, where d_h is the number of hidden neurons. Let $\mathbf{u} \in \mathbb{R}^{d_h}$ be the activation of the penultimate layer. The output is computed as follow

$$\begin{aligned} \mathbf{a} &= W\mathbf{u} \\ \hat{\mathbf{y}} &= \delta(\mathbf{a}) \end{aligned}$$

where δ is the softmax output function. The derivative of the loss at $\mathbf{z}$ w.r.t. W is

$$\frac{\partial \ell(\mathbf{z})}{\partial W} = \nabla_{\mathbf{a}} \ell(\mathbf{z}) \, \mathbf{u}^\top \tag{3}$$

$$= \begin{bmatrix} \nabla_{\mathbf{a}} \ell(\mathbf{z})_1 \mathbf{u}^\top \\ \vdots \\ \nabla_{\mathbf{a}} \ell(\mathbf{z})_{d_y} \mathbf{u}^\top \end{bmatrix} \tag{4}$$

The gradient $\nabla_{\mathbf{a}} \ell(\mathbf{z})$ is

$$(\nabla_{\mathbf{a}} \ell)^\top = \frac{\partial \ell}{\partial \mathbf{a}} \tag{5}$$

$$= \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{a}} \tag{6}$$

$$= \begin{bmatrix} \frac{\partial \ell}{\partial \hat{y}_1} & \cdots & \frac{\partial \ell}{\partial \hat{y}_k} & \cdots & \frac{\partial \ell}{\partial \hat{y}_{d_y}} \end{bmatrix} \times \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial a_1} & \frac{\partial \hat{y}_1}{\partial a_2} & \cdots & \frac{\partial \hat{y}_1}{\partial a_{d_h}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \hat{y}_k}{\partial a_1} & \frac{\partial \hat{y}_k}{\partial a_2} & \cdots & \frac{\partial \hat{y}_k}{\partial a_{d_h}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \hat{y}_{d_y}}{\partial a_1} & \frac{\partial \hat{y}_{d_y}}{\partial a_2} & \cdots & \frac{\partial \hat{y}_{d_y}}{\partial a_{d_h}} \end{bmatrix} \tag{7}$$

$$= \begin{bmatrix} \frac{\partial \ell}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial a_1} & \cdots & \frac{\partial \ell}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial a_k} & \cdots & \frac{\partial \ell}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial a_{d_h}} \end{bmatrix} \tag{8}$$

We go from Eqn. 7 to Eqn. 8 by using the following fact

$$\frac{\partial \ell}{\partial \hat{y}_i} = \begin{cases} 0 \text{ if } i \neq k \\ \frac{1}{\hat{y}_i} \text{ if } i = k \end{cases}$$

We also have

$$\frac{\partial \hat{y}_k}{\partial a_i} = \begin{cases} \hat{y}_k(1 - \hat{y}_k) \text{ if } i = k \\ -\hat{y}_k \hat{y}_i \text{ if } i \neq k \end{cases}$$

Substitute this into Eqn. 8 we have

$$\nabla_{\mathbf{a}} \ell = \begin{bmatrix} -\hat{y}_1 \\ \vdots \\ 1 - \hat{y}_k \\ \vdots \\ -\hat{y}_{d_y} \end{bmatrix}$$

Because $1 - \hat{y}_k = \sum_{j \neq k} \hat{y}_j$, $1 - \hat{y}_k$ is much greater than $\hat{y}_j$ in general. Substitute this into Eqn. 4, we see that the magnitude of the k-th row is much larger than than of other rows. We also note that the update for the k-th row of W has the opposite direction of the updates for other rows.

Let's consider the inner product of the gradients of two data points $\mathbf{z}$ and $\mathbf{z}'$ with label k and k'. Let's consider the case where $k' \neq k$ first.

$$\text{vec}\left(\frac{\partial \ell(\mathbf{z})}{\partial W}\right)^\top \text{vec}\left(\frac{\partial \ell(\mathbf{z}')}{\partial W}\right) = (\nabla_{\mathbf{a}} \ell^\top \nabla_{\mathbf{a}'} \ell)(\mathbf{u}^\top \mathbf{u}') \tag{9}$$

Intuitively, the product $\nabla_{\mathbf{a}} \ell^\top \nabla_{\mathbf{a}'} \ell$ is small because the large element $\nabla_{\mathbf{a}} \ell_k = 1 - \hat{y}_k$ is multiplied to the small element $\nabla_{\mathbf{a}'} \ell_k = \hat{y}'_k$ and the large element $\nabla_{\mathbf{a}'} \ell_{k'} = 1 - \hat{y}'_{k'}$ is multiplied to the small element $\nabla_{\mathbf{a}} \ell_{k'} = \hat{y}_{k'}$. To make it more concrete, let's assume that $\hat{y}_k = \alpha \approx 1$ and $\hat{y}_i = \frac{1-\alpha}{d_y - 1} = \beta$ for $i \neq k$. We assume the same condition for $\mathbf{y}'$.

$$\begin{aligned} \nabla_{\mathbf{a}} \ell^\top \nabla_{\mathbf{a}'} \ell &= (\hat{y}_k - 1)\hat{y}'_k + (\hat{y}'_{k'} - 1)\hat{y}_{k'} + \sum_{i=1, i \neq k, k'}^{d_y} \hat{y}_i \hat{y}'_i \\ &= (d_y - 2)\beta^2 - 2(d_y - 1)\beta^2 \\ &= -d_y \beta^2 \\ &= -\frac{d_y(1-\alpha)^2}{(d_y - 1)^2} \end{aligned} \tag{10}$$

$\alpha \approx 1$ implies $1 - \alpha \approx 0$ and $\beta \approx 0$. Eqn. 10 implies that as the model is more confident about the label of $\mathbf{z}$ and $\mathbf{z}'$, the product $\nabla_{\mathbf{a}} \ell^\top \nabla_{\mathbf{a}'} \ell$ tends toward 0 with a quadratic rate.

The sign of the gradient product depends on the sign of $\nabla_{\mathbf{a}} \ell^\top \nabla_{\mathbf{a}'} \ell$ and $\mathbf{u}^\top \mathbf{u}'$. The signs of $\nabla_{\mathbf{a}} \ell^\top \nabla_{\mathbf{a}'} \ell$ and $\mathbf{u}^\top \mathbf{u}'$ are random variables that depend on the noise in the features $\mathbf{u}$ and $\mathbf{u}'$ and the weight matrix W. If the model $f_{\boldsymbol{\theta}}$ cannot learn a good representation of the input then the feature $\mathbf{u}$ and the sign of $\mathbf{u}^\top \mathbf{u}'$ could be very noisy. $\text{sign}(\mathbf{u}^\top \mathbf{u}')$ is even noisier if $\mathbf{z}$ and $\mathbf{z}'$ are from different classes.

We now consider the case where $k' = k$. When $k' = k$, $\nabla_{\mathbf{a}} \ell^\top \nabla_{\mathbf{a}'} \ell$ is always positive. The sign of the gradient product only depends on $\mathbf{u}^\top \mathbf{u}'$. That explains why the product of gradients of data points from the same class is much less noisy.

Furthermore, the magnitude of $\nabla_{\mathbf{a}} \ell^\top \nabla_{\mathbf{a}'} \ell$ is larger than that in the case $k' \neq k$ because the large element $1 - \hat{y}_k$ is multiplied to the large element $1 - \hat{y}'_k$. More concretely, under the same assumption as in the case $k' \neq k$, we have

$$\begin{aligned}\nabla_{\mathbf{a}} \ell^\top \nabla_{\mathbf{a}'} \ell &= (1 - \hat{y}_k)(1 - \hat{y}'_k) + \sum_{i=1, i \neq k}^{d_y} \hat{y}_i \hat{y}'_i \\ &= (1 - \alpha)^2 + (d_y - 1)\beta^2 \end{aligned} \tag{11}$$

From Eqn. 11, we see that when $k' = k$, the magnitude of $\nabla_{\mathbf{a}} \ell^\top \nabla_{\mathbf{a}'} \ell$ is approximately d_y times larger than that when $k' \neq k$.

E Unpublished manuscript

We attach the unpublished manuscript from (Anonymous, 2022) here for reference.

A paper under review

Anonymous

Abstract

Errors in datasets could significantly damage the performance and robustness of machine learning models trained on these datasets. Error detection and correction is a challenging problem, especially for big datasets and deep models. In this paper, we show that influence function and its variants can detect errors in large-scale deep learning datasets. However, they are unstable, computationally expensive, and depend on strong assumptions. We develop a high-performance, stable, scalable, data-efficient error detection algorithm that consistently outperforms influence functions. Our method uses a supervised learning model trained on a small clean dataset to detect mislabeled data instances in a much larger dataset. The model can achieve high error detection accuracy even when the supervised learning task's accuracy is low. We analyze our error detection algorithms and discuss directions for further improvements.

1 Introduction

Deep learning (DL) (Schmidhuber, 2015; LeCun et al., 2015) is currently the dominant approach to Artificial Intelligence. Although the performance of unsupervised deep learning has improved significantly in the last few years, supervised deep learning usually has better performance and requires fewer data and computing resources. However, the performance and robustness of supervised models could be significantly damaged by bad training examples. In recent years, the data-centric AI approach (c.f. Russakovsky, 2021), a principled approach to building better AI systems by improving the quality of the data, has garnered increasing attention from the research community and the industry. Following the data-centric AI approach, we propose algorithms for improving the quality of large-scale supervised learning datasets.

Bad training examples could be the result of mistakes in the labeling process or the outliers in the data. Human error rate could range from a few to tens of percent on large datasets. For example, average human's top-5 error rate on the ImageNet (Deng et al., 2009) dataset is 5.1%. For more challenging datasets where expert knowledge is required, the error rate could be even higher. A correctly labeled data point can still be an outlier and have a bad effect on the model. For brevity, we call these bad data points errors. Correcting or removing errors improves the performance and robustness of models trained on that data. Unfortunately, automatic error detection and correction is a non-trivial task, especially for large datasets and deep models. The large scale of modern datasets makes manual inspection of the data prohibitively expensive. And the immense complexity of deep models makes it extremely hard to reliably and efficiently predict the effect a data point has on the performance of a deep model (Basu et al., 2021).

In this paper, we develop algorithms for detecting bad examples in large-scale deep learning datasets. We focus on datasets for classification problems but our methods can be generalized to other supervised learning datasets. We start by adapting gradient-based influence functions (Koh and Liang, 2017; Charpiat et al., 2019; Pruthi et al., 2020; Barshan et al., 2020), a class of algorithms that can estimate the influence of a data point on the performance of a deep model, to the error detection problem (Sec. 2). If a data point has a bad influence on the model, it is considered an error. Our experiments show that although these algorithms can detect errors, they are unstable and computationally expensive. We explain the instability and the unsatisfactory performance of these algorithms in Sec. 3.1.

In Sec. 3, we propose a data-efficient supervised learning algorithm for error detection. Our algorithm uses a supervised learning model trained on a small clean dataset to identify potentially bad examples in a much larger dataset. Our algorithm is faster, more stable, and has higher performance than gradient-based algorithms. Our method could use as few as 500 clean training examples and still significantly outperforms gradient-based methods.

Contributions

1. A way to adapt influence functions to the error detection problem.
2. A high performance, stable, scalable, data efficient error detection algorithm.
3. An explanation for the success and failure of error detection algorithms.

2 Background and related works

2.1 Problem formulation

We define the notations used in this paper. A data point $\mathbf{z}^{(i)} = \left(\mathbf{x}^{(i)},\ \mathbf{y}^{(i)}\right)$ is a pair of input $\mathbf{x}^{(i)} \in \mathcal{X}$ and output $\mathbf{y}^{(i)} \in \mathcal{Y}$. A dataset of n distinct data points is denoted as $\mathcal{Z} = \left\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(n)}\right\}$. Let's denote $\mathcal{Z}_{-i} = \mathcal{Z} \setminus \mathbf{z}^{(i)}$. A machine learning model is a parametric function of the form $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$. The loss of $f_{\boldsymbol{\theta}}$ on $\mathcal{Z}$ is defined as

$$\mathcal{L}_{\mathcal{Z},\boldsymbol{\theta}} = \frac{1}{n}\sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}) \tag{1}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{z}^{(i)}; \boldsymbol{\theta}) \tag{2}$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ is the loss function. When it is clear from the context, we write $\ell(\mathbf{z})$ instead of $\ell(\mathbf{z}; \boldsymbol{\theta})$. Let's denote $\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{Z},\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\theta}}_{-i} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{Z}_{-i},\boldsymbol{\theta}}$. The influence of a data point $\mathbf{z}^{(i)}$ on the model $f_{\boldsymbol{\theta}}$ is defined as

$$s^{(i)} = \mathcal{L}_{\mathcal{Z}_{-i},\hat{\boldsymbol{\theta}}_{-i}} - \mathcal{L}_{\mathcal{Z},\hat{\boldsymbol{\theta}}} \tag{3}$$

$$\approx \frac{1}{n}\sum_{j=1}^{n} \ell(\mathbf{z}^{(j)}; \hat{\boldsymbol{\theta}}_{-i}) - \ell(\mathbf{z}^{(j)}; \hat{\boldsymbol{\theta}}) \tag{4}$$

$s^{(i)} < 0$ means that removing $\mathbf{z}^{(i)}$ from $\mathcal{Z}$ decreases the loss. $\mathbf{z}^{(i)}$ is bad for $f_{\boldsymbol{\theta}}$ if $s^{(i)} < 0$. Given a dataset $\mathcal{Z}$, we have to detect bad data points in $\mathcal{Z}$. Correcting/removing these data points results in a new dataset $\mathcal{Z}^*$. Let $f_{\boldsymbol{\theta}^*}$ be the model trained on $\mathcal{Z}^*$. $f_{\boldsymbol{\theta}^*}$ is expected to perform better than $f_{\hat{\boldsymbol{\theta}}}$.

An error detection model $g_{\boldsymbol{\phi}} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ assigns a score to each data point in $\mathcal{Z}$. In this paper, a lower score indicates that the data point is more likely an error. The data points are ranked based on their scores. The most suspicious points are re-examined by humans or are removed from $\mathcal{Z}$.

2.2 Gradient based error detection

Algorithm 1 Gradient based error detection

```
Require:
1: $\mathcal{Z} = \left\{\mathbf{z}^{(i)}\right\}_{i=1}^{n}$: a big noisy dataset
2: $\mathcal{Z}' = \left\{\mathbf{z}'^{(j)}\right\}_{j=1}^{m}$: a small clean dataset
3: $f_{\hat{\boldsymbol{\theta}}}$: a deep model pretrained on $\mathcal{Z}$
4: $\mathrm{sim}(\cdot,\cdot)$: a similarity measure
Ensure: $\hat{\mathcal{Z}}$: data points in $\mathcal{Z}$ ranked by score
5: for $\mathbf{z}^{(i)} \in \mathcal{Z}$ do
6:     $s^{(i)} = \frac{1}{m}\sum_{j=1}^{m} \mathrm{sim}(\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)}), \nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}'^{(j)}))$
7: end for
8: $\hat{\mathcal{Z}} = \mathrm{sort}(\mathcal{Z}, \mathrm{key} = \boldsymbol{s}, \mathrm{ascending} = \mathrm{True})$
9: return $\hat{\mathcal{Z}}$
```

Naive application of Eqn. 3 to the error detection problem is intractable as it requires retraining the model $f_{\boldsymbol{\theta}}$ on every subset $\mathcal{Z}_{-i},\ i = 1, ..., n$. Koh and Liang (2017) proposed the influence function (IF) to efficiently estimate the influence of a data point on other data points without retraining. IF estimates the change in loss of a data point $\mathbf{z}^{(j)}$ when a data point $\mathbf{z}^{(i)}$ is removed from $\mathcal{Z}$, as follows

$$IF(\mathbf{z}^{(i)}, \mathbf{z}^{(j)}) = \ell(\mathbf{z}^{(j)}; \hat{\boldsymbol{\theta}}_{-i}) - \ell(\mathbf{z}^{(j)}; \hat{\boldsymbol{\theta}}) \tag{5}$$

$$\approx \frac{1}{n}\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)}; \hat{\boldsymbol{\theta}})^\top H_{\hat{\boldsymbol{\theta}}}^{-1} \nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(j)}; \hat{\boldsymbol{\theta}}) \tag{6}$$

where $H_{\hat{\boldsymbol{\theta}}} = \partial^2\mathcal{L}_{\mathcal{Z},\hat{\boldsymbol{\theta}}}/\partial\hat{\boldsymbol{\theta}}^2$ is the Hessian matrix. Averaging Eqn. 6 over $\mathcal{Z}$ gives an estimate of $s^{(i)}$ (Eqn. 4). Koh and Liang (2017) require H to be positive definite but empirical evidence shows that IF is still a reasonable estimate when this requirement is not satisfied. The constant factor $1/n$ in Eqn. 6 does not affect the output of our algorithms, we remove it to avoid cluttering.

From Eqn. 6, we see that IF is a similarity measure between $\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)})$ and $\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(j)})$. Several variants of IF with faster similarity measures have been proposed:

Grad-Dot (GD) (Charpiat et al., 2019) is much faster than IF as it does not require computing and inverting the Hessian

$$GD(\mathbf{z}^{(i)}, \mathbf{z}^{(j)}) = \left\langle \nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)}), \nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(j)}) \right\rangle \tag{7}$$

Grad-Cos (GC) (Charpiat et al., 2019) improves upon GD by normalizing the gradients

$$GC(\mathbf{z}^{(i)}, \mathbf{z}^{(j)}) = \cos\left(\nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(i)}), \nabla_{\hat{\boldsymbol{\theta}}}\ell(\mathbf{z}^{(j)})\right) \tag{8}$$

Tracing Gradient Descent (TracIn) (Pruthi et al., 2020) improves the estimation precision by summing the influence over the training process

$$TracIn(\mathbf{z}^{(i)}, \mathbf{z}^{(j)}) = \sum_{t=1}^{T} \eta_t \left\langle \nabla_{\boldsymbol{\theta}^{(t)}} \ell(\mathbf{z}^{(i)}), \nabla_{\boldsymbol{\theta}^{(t)}} \ell(\mathbf{z}^{(j)}) \right\rangle \quad (9)$$

where T is the number of checkpoints, η_t is the learning rate, and $\boldsymbol{\theta}^{(t)}$ is the parameter at checkpoint t.

Gradient-based algorithms have high complexity. The number of calls to the scoring function sim() function is nm. The main computational overhead of gradient-based algorithms comes from this similarity function.

We explain how influence functions can be used for error detection. Recall that the gradient descent update rule for a parametric model $f_{\boldsymbol{\theta}}$ is

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{Z}} \quad (10)$$

$$= \boldsymbol{\theta}^{(t)} - \frac{\eta}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \ell(\mathbf{z}^{(i)}) \quad (11)$$

If the data is clean and the learning rate η is small enough, then moving $\boldsymbol{\theta}$ in the direction of $-\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{Z}}$ will decrease the loss. An errorneous data point $\mathbf{z}$ is likely to have bad influence on the model, i.e. moving $\boldsymbol{\theta}$ in the direction of $-\nabla_{\boldsymbol{\theta}} \ell(\mathbf{z})$ is likely to increase the loss. In other words, the direction of the gradient of an erroneous data point should be opposite to that of correct data points. Fig. 1 demonstrates this idea on a simple 2D dataset with a linear classifier.

We use this intuition to develop a data-efficient algorithm for error detection. Alg. 1 requires a small clean dataset $\mathcal{Z}'$ and a model $f_{\hat{\boldsymbol{\theta}}}$ trained on the large, noisy dataset $\mathcal{Z}$.[1] The clean data points in $\mathcal{Z}'$ are selected manually at random. We compute the gradient of every data point $\mathbf{z}^{(i)} \in \mathcal{Z}$ and compare it to the gradient of clean data points $\mathbf{z}'^{(j)} \in \mathcal{Z}'$ using one of the above similarity measures. Because gradients could be noisy (Sec. 3.1), we average the similarity score over $\mathcal{Z}'$ to make it more stable. The output $\hat{\mathcal{Z}}$ is sorted in descending order of the suspiciousness, i.e. the first example in $\hat{\mathcal{Z}}$ is the most likely to be an error.

[1]We trained $f_{\boldsymbol{\theta}}$ on the small clean dataset $\mathcal{Z}'$ and used that model to detect errors in $\mathcal{Z}$. Tab. 4 in Appendix C shows that this algorithm has lower detection accuracy than Alg. 1.

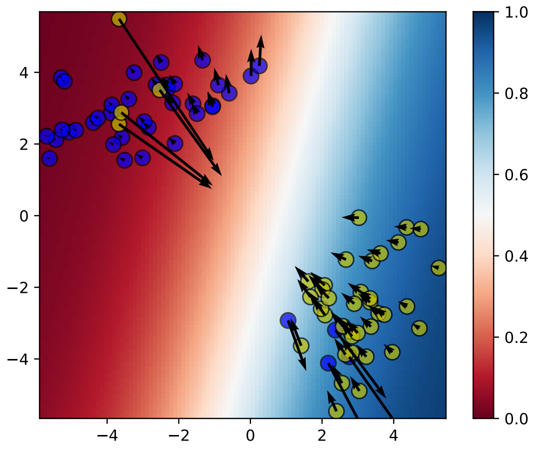


Figure 1: Gradient based error detection. The arrow at each data point shows the gradient of the loss at that data point w.r.t. the parameter. Gradients of correctly classified examples point in roughly the same direction. Gradients of wrongly classified examples point in the opposite direction.

2.3 Feature based error detection

For the sake of completeness, we try to use features for error detection as they are commonly used in influence estimation (Hanawa et al., 2021; Pezeshkpour et al., 2021). Based on Alg. 1, we replace the gradient $\nabla_{\boldsymbol{\theta}} \ell(\mathbf{z})$ with the feature $\sigma(\mathbf{x})$ and use negative Euclidean distance (EUC), dot product (DOT), and cosine similarity (COS) as similarity measures. However, our experiments show that feature-based error detection is not effective (Sec. 4). This is because the feature $\sigma(\mathbf{x})$ is a function of the input $\mathbf{x}$ only,[2] it does not contain information about the label $\mathbf{y}$.

2.4 Other error detection methods

Previous works on error detection can be categorized into 3 categories: rule-based, statistics-based, and machine learning-based approaches.

The rule-based approach (Chu et al., 2013) is commonly used for structured data. This is not suitable for deep learning as the rules in many deep learning datasets are not easy to find and describe.

The statistics-based approach (Huang and He, 2018) is also commonly used for structured data such as tabular data. This approach exploits the statistical dependencies between features to perform error detection and correction. Several papers combine machine learning techniques such as data augmentation (Pham et al., 2021), few-shot learning (Heidari et al., 2019), active learning (Neutatz

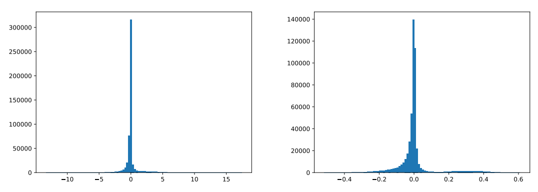
[2]Although $\mathbf{y}$ can indirectly affect $\sigma(\mathbf{x})$ through back-propagation, we empirically found this effect very subtle and noisy.

et al., 2019) with rule based and statistics based approach to detect/correct errors in tabular data.

The machine learning-based approach is more general than the previous two approaches as it makes fewer or no assumptions about the data. Krishnan et al. (2016) combines active learning and convex models to detect errors on several small classification datasets. Because the algorithm requires the model to be convex, it is not applicable to deep networks. Our algorithms work for all types of deep networks and are scalable. Furthermore, our gradient-based algorithms require no human intervention, and our model-based algorithm can be modified to remove the need for humans (Sec. 4.4).

3 Model based error detection

3.1 Motivation



(a) GD (b) GC

Figure 2: Distribution of GD, GC scores over the Ohsumed dataset. GD, GC were computed using last layer gradient of a BERT model (Devlin et al., 2019). The similarity scores concentrate around 0. Similar pattern is observed on other datasets used in this paper.

Our experiments show that gradient-based error detection algorithms are unstable and feature-based algorithms do not work in general. This section explains the reason for such behaviors and motivates our model-based error detection algorithm.

Because computing the gradient of the loss of every data point w.r.t. the whole network is prohibitively expensive, the gradient w.r.t. the last layer is usually used in practice. Pezeshkpour et al. (2021) empirically verified that influence functions with last layer gradient perform almost as well as influence functions with full gradient. They also observed that GD and IF are highly correlated with a Spearman Correlation score higher than 0.95. Therefore, we analyze GD with last layer gradient and generalize the result to other variants of IF.

Let's consider a L layer network that receives input $\mathbf{x} \in \mathbb{R}^{d_0}$ and produces output $\hat{\mathbf{y}} \in \mathbb{R}^{d_L}$. Let $\mathbf{u} \in \mathbb{R}^{d_{L-1}}$ be the output of layer $L-1$, $W \in \mathbb{R}^{d_y \times d_{L-1}}$ be the parameters of the last layer, δ be the output function. For simplicity, we remove the bias term from the last layer. Let $\mathbf{a} = W\mathbf{u}$, $\hat{\mathbf{y}} = \delta(\mathbf{a})$, and $\ell = \ell(\hat{\mathbf{y}}, \mathbf{y}; W)$.

$$\begin{aligned}\frac{\partial \ell}{\partial W} &= \frac{\partial \ell}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial W} \\ &= \nabla_{\mathbf{a}} \ell \, \mathbf{u}^\top\end{aligned}$$

Let $\nabla_W \ell = \mathrm{vec}(\partial \ell / \partial W)$ be the vectorization of the derivative w.r.t. W. The dot product of the gradients of $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ and $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)})$ is

$$\nabla_W \ell^{(i)\top} \nabla_W \ell^{(j)} = (\nabla_{\mathbf{a}} \ell^{(i)\top} \nabla_{\mathbf{a}} \ell^{(j)})(\mathbf{u}^{(j)\top} \mathbf{u}^{(i)})$$

Ideally, a negative GD score should mean that similar features $\mathbf{u}^{(i)}$ and $\mathbf{u}^{(j)}$ are mapped to different outputs $\mathbf{a}^{(i)}$ and $\mathbf{a}^{(j)}$. In practice, the dot product between features could be noisy because it depends on the quality of the feature $\mathbf{u}$. If the network is not well trained, then $\mathbf{u}$ could be arbitrarily distributed, and $\mathbf{u}^{(j)\top} \mathbf{u}^{(i)}$ could be uninformative. If the network is well trained then the features of different classes tend to be pushed into different orthogonal subspaces of $\mathbb{R}^{d_{L-1}}$ (Csordás et al., 2021). As a result, $\mathbf{u}^{(j)\top} \mathbf{u}^{(i)}$ is close to 0 for $\mathbf{y}^{(i)} \neq \mathbf{y}^{(j)}$ (Fig. 2), and a tiny noise could flip the sign of $\mathbf{u}^{(j)\top} \mathbf{u}^{(i)}$. These factors make gradient-based error detection less effective.

We aim to develop an algorithm that does not depend on the noisy feature $\mathbf{u}$. A solution is to use a supervised model $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$. For every input $\mathbf{x}^{(i)}$, compare the model's output $\hat{\mathbf{y}}^{(i)} = f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ to the target output $\mathbf{y}^{(i)}$. If $\hat{\mathbf{y}}^{(i)}$ is largely different from $\mathbf{y}^{(i)}$ then $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ is likely an error. This solution has several problems:

1. The detection accuracy depends on the quality of the supervised model $f_{\boldsymbol{\theta}}$.
2. If $f_{\boldsymbol{\theta}}$ is trained on $\mathcal{Z}$ then it might overfit to the errors in $\mathcal{Z}$.

Collecting a large clean dataset for training $f_{\boldsymbol{\theta}}$ is infeasible as it is prone to error and is as expensive as manually cleaning $\mathcal{Z}$. In the next section, we develop a data efficient solution to this problem.

3.2 Method

Our model-based error detection algorithm is presented in Alg. 2. Besides the large noisy dataset $\mathcal{Z}$ and the small clean dataset $\mathcal{Z}'$, our algorithm requires a validation set $\mathcal{Z}^+$. $\mathcal{Z}^+$ is used for estimating the error detection performance of $f_{\hat{\boldsymbol{\theta}}}$. $\mathcal{Z}^+$

is created by selecting a small set of clean examples and randomly corrupting the output of $p\%$ of these examples. We note that the distribution of errors in $\mathcal{Z}$ might not be uniform. The mismatch between the error distributions in $\mathcal{Z}$ and $\mathcal{Z}^+$ could lower the performance of our algorithm. We discuss this issue in Sec. 4. The sizes of $\mathcal{Z}'$ and $\mathcal{Z}^+$ are hyper-parameters. Our experiments show that our algorithm works well even when $\mathcal{Z}'$ and $\mathcal{Z}^+$ are hundreds to thousands of times smaller than $\mathcal{Z}$.

Our algorithm trains $f_{\boldsymbol{\theta}}$ for T iterations. At iteration t, $f_{\hat{\boldsymbol{\theta}}^{(t-1)}}$ is trained on $\mathcal{Z}'$ for u epochs to get $f_{\hat{\boldsymbol{\theta}}^{(t)}}$. The error detection accuracy of $f_{\hat{\boldsymbol{\theta}}^{(t)}}$ is measured on $\mathcal{Z}^+$. The function errDetectAcc ranks examples in $\mathcal{Z}^+$ by their score s (line 24 in Alg. 2) and counts the number of errors in the top $p\%$ of $\mathcal{Z}^+$. In our experiments, we fix $p = 50\%$. The model with the best error detection accuracy is retained and used for detecting errors in $\mathcal{Z}$ (line 23-26 in Alg. 2). An example $\mathbf{z}$ is likely an error if its loss $\ell(f_{\boldsymbol{\theta}^*}(\mathbf{x}), \mathbf{y})$ is high. To make it consistent with Alg. 1, we sort the examples in $\mathcal{Z}$ in ascending order of the negative of the loss.

The initial clean dataset $\mathcal{Z}'^{(1)}$ might be too small and does not cover all regions of the data space. We use active learning (Settles, 2009) to gradually expand the clean dataset $\mathcal{Z}'$ (line 16-21 in Alg. 2). We use an information measure $\mathrm{I}(\cdot)$ to find data points that are the most informative to the model $f_{\hat{\boldsymbol{\theta}}^{(t)}}$. The top r most informative data points are cleaned and added to $\mathcal{Z}'^{(t)}$. For classification datasets, we empirically find that the entropy measure (Shannon, 2001) is the most stable and performant. The entropy of a data point $\mathbf{z}$ under the classification model $f_{\boldsymbol{\theta}}$ is

$$\mathrm{Ent}(\mathbf{z}; \boldsymbol{\theta}) = \sum_{i=1}^{C} f_{\boldsymbol{\theta}}(\mathbf{x})_i \log\left(f_{\boldsymbol{\theta}}(\mathbf{x})_i\right)$$

where C is the number of classes in the dataset. $\mathrm{Ent}(\mathbf{z})$ is maximized if the model assign the same probability to all of the C classes. Intuitively, our algorithm adds to $\mathcal{Z}'$ examples that confuse the model the most. This way, the set $\mathcal{Z}'$ will be the most diverse (according to the model $f_{\boldsymbol{\theta}}$, and the performance of $f_{\boldsymbol{\theta}}$ will be improved the fastest. For non-classification datasets, we can use a distance or divergence measure between $\hat{\mathbf{y}}$ and $\mathbf{y}$ as the criterion for selecting informative examples. Selected data points are cleaned manually (line 20 in Alg. 2). In Sec. 4.4 we discuss methods to automate this step.

Algorithm 2 Model based error detection

```
Require:
 1: $\mathcal{Z} = \{\mathbf{z}^{(i)}\}_{i=1}^{n}$: a big noisy dataset
 2: $\mathcal{Z}' = \{\mathbf{z}'^{(j)}\}_{j=1}^{m}$: a small clean dataset
 3: $\mathcal{Z}^+ = \{\mathbf{z}^{+(k)}\}_{k=1}^{q}$: a validation set
 4: $f_{\boldsymbol{\theta}}$: a deep model
 5: $T$: No. iterations
 6: $u$: No. epochs for train() function
 7: $r$: No. additional points per iteration
Ensure: $\hat{\mathcal{Z}}$: data points in $\mathcal{Z}$ ranked by score
 8: Initialize $f_{\boldsymbol{\theta}^{(0)}}$, $\mathcal{Z}'^{(1)} = \mathcal{Z}'$
 9: for $t = 1, ..., T$ do
10:     $f_{\hat{\boldsymbol{\theta}}^{(t)}} = \mathrm{train}(f_{\hat{\boldsymbol{\theta}}^{(t-1)}}, \mathcal{Z}'^{(t)}, u)$
11:     $P_{\hat{\boldsymbol{\theta}}^{(t)}} = \mathrm{errDetectAcc}(f_{\hat{\boldsymbol{\theta}}^{(t)}}, \mathcal{Z}^+, p)$
12:     if $P_{\hat{\boldsymbol{\theta}}^{(t)}} > P_{\boldsymbol{\theta}^*}$ then
13:         $\boldsymbol{\theta}^* = \hat{\boldsymbol{\theta}}^{(t)}$
14:         $P_{\boldsymbol{\theta}^*} = P_{\hat{\boldsymbol{\theta}}^{(t)}}$
15:     end if
16:     for $\mathbf{z}^{(i)} \in \mathcal{Z} \setminus \mathcal{Z}'^{(t)}$ do
17:         $\mathrm{I}(\mathbf{z}^{(i)}) = \mathrm{infoMeasure}(\mathbf{z}^{(i)})$
18:     end for
19:     $\mathcal{U} = \{r$ examples with highest I in $\mathcal{Z} \setminus \mathcal{Z}'^{(t)}\}$
20:     $\mathcal{U} = \mathrm{clean}(\mathcal{U})$          ▷ requires human
21:     $\mathcal{Z}'^{(t+1)} = \mathcal{Z}'^{(t)} \cup \mathcal{U}$
22: end for
23: for $\mathbf{z}^{(i)} \in \mathcal{Z}$ do
24:     $s^{(i)} = -\ell(f_{\boldsymbol{\theta}^*}(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$
25: end for
26: $\hat{\mathcal{Z}} = \mathrm{sort}(\mathcal{Z}, \mathrm{key} = \boldsymbol{s}, \mathrm{ascending} = \mathrm{True})$
27: return $\hat{\mathcal{Z}}$
```

The size of the gradient is equal to the number of parameters in the network which is thousands or millions of times larger than the number of outputs of the network. Therefore, using the loss ℓ as the scoring function is computationally more efficient than using similarity functions in Sec. 2.2. Furthermore, our model-based algorithm makes only n calls to the scoring function ℓ. Compare to gradient-based algorithms, model-based algorithm makes m times fewer calls to the scoring function.

If $\mathcal{Z}$ is a classification dataset, then $\mathbf{y}$ is a one hot vector and $\hat{\mathbf{y}}$ is a distribution over C classes. If ℓ is the cross entropy loss then $\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{C} y_i \log \hat{y}_i = -\log \hat{y}_k$ where k is the label of the example $(\mathbf{x}, \mathbf{y})$. The lower $\hat{y}_k$ is, the higher the loss $\ell(\hat{\mathbf{y}}, \mathbf{y})$ and the more likely the example is an error. So for classification dataset with cross entropy loss, instead of computing and ranking examples based on the loss, we can simply rank the examples by the probability of the label.

4 Experiments

4.1 Datasets and models

We conduct experiments on a variety of models and datasets to verify our algorithms. We select 4 classification datasets in the natural language processing and computer vision domain (Tab. 1, Appendix A). The original datasets contain no noise so to test our algorithms, we add synthetic noise to the data. In our experiments, we use 2 types of synthetic noise:

1. *Random noise*: we change the label of a data point to another random class.

2. *Structured noise*: we create a rule h that maps the label of data points in a class to another fixed class. The label selected instances in class i is changed to $h(i)$. The map h satisfies the following condition $h(i) \neq h(j)\ \forall\ i, j = 1, ..., C$ and $i \neq j$.

The structured noise models the situation where human labelers make systematic errors. For example, a human labeler does not know the difference between a giant schnauzer and a standard schnauzer (2 classes in the ImageNet dataset) and uses the same label for these two breeds of dogs. We note that there are infinitely many ways to mislabel data. We present here two representative types of noise and use them to study the effect of noise distribution on error detection algorithms.

To create the large noisy dataset $\mathcal{Z}$, for each dataset in Tab. 1, we randomly select 20% of the training set of each dataset and change the labels of selected data points using one of the above methods. The clean dataset $\mathcal{Z}'$ is created by drawing without replacement 500 random samples from the test dataset. To create $\mathcal{Z}^+$, the validation set for error detection, we draw without replacement 200 random samples from the test set and randomly change the label of 100 samples in that dataset.

4.2 Error detection on deep learning datasets

In this section, we test the performance of error detection algorithms (Tab. 2). Details about the training procedure and hyper-parameters are deferred to Appendix B. For each algorithm, we select the top $k\%$ of the output dataset $\hat{\mathcal{Z}}$ and compute the percentage of errors in that top $k\%$. In Tab. 2, k is varied from 5% to 20%. We note that random guessing should have an error detection accuracy of approximately $q = 20\%$ for all values of k. Tab. 2 reports the averaged result of 4 different runs of these algorithms.

The first thing to note is that the error detection accuracy decreases as k increases. This is expected as the suspiciousness decreases as we go down the sorted list $\hat{\mathcal{Z}}$.

For most datasets and types of noise, EUC performs worse than random guessing. DOT's performance varies greatly from dataset to dataset. DOT outperforms other feature-based and gradient-based algorithms on ImageNet100 dataset with random noise but its performance on other datasets is similar to random guessing. COS is the most stable feature-based algorithm. However, its performance is only slightly better than random guessing. We conclude that feature-based algorithms are not effective.

Gradient-based algorithms have better performance than feature-based algorithms and random guessing. However, the performance of gradient-based algorithms still fluctuates when the dataset and/or random seed change. This behavior confirms our conclusion in Sec. 3.1 that the performance of gradient-based algorithms is sensitive to We observe that the performance of IF, GD, and GC are pretty similar. TracIn behaves more erratically across datasets. It outperforms the model-based algorithm on the Oshumed dataset with structured noise but on other datasets, it is dominated by the model-based and other gradient-based algorithms. A reason for this instability is that TracIn accumulates the gradient dot product over the training process and this gradient dot product could be very noisy in the early phase of the training process.

Our model-based algorithm has the best performance on almost all datasets and types of noise and is the most stable algorithm. Our algorithm is also less sensitive to changes in random seed as it often has the lowest standard deviation. This is because our model-based algorithm does not depend on the noisy feature dot product.

We observe that the model's error detection accuracy is significantly higher than its classification accuracy (Tab. 2 and Tab. 5). In line 24 of Alg. 2, we see that the score of a data point $(\mathbf{x}, \mathbf{y})$ is simply the negative of its loss. The loss is high when the model assigns a low probability to the (noisy) label $\mathbf{y}$ and it can do so without assigning the highest probability to the correct label. Thus, the model can detect an erroneous example without classifying it correctly.

Finally, we discuss the effect of noise distribution on error detection algorithms. The error detection accuracy of gradient-based algorithms on structured noise is higher than that on random noise. One reason for this behavior is that the machine learning model f_{θ} in gradient-based algorithms has adapted to the patterns in structured noise so it can detect noisy examples more effectively. This is an advantage of machine learning-based error detection over rule-based and statistics-based algorithms. More interestingly, the same behavior is observed for our model-based algorithm. At first glance, one might expect the performance of model-based algorithm on structured noise to be lower than that on random noise because the model was trained to detect uniform random noise on the validation set $\mathcal{Z}^+$. We explain this phenomenon as follows. The uniform random distribution is the maximal entropy distribution. The entropy of uniform random noise is higher than that of structured noise. Therefore, in general, detecting random noise is a harder problem than detecting structured noise. The model f_{θ} was trained on a harder problem (detecting random noise in $\mathcal{Z}^+$) and was tested on an easier problem (detecting structured noise in $\mathcal{Z}$). As a result, the performance on structured noise is higher. This explanation applies to gradient-based algorithms as well.

4.3 Improving datasets with error correction

We use our error detection algorithms to rank examples in $\mathcal{Z}$ and correct/remove the top 10% most suspicious examples. We observe a significant improvement in the classification performance of models trained on the new datasets (Tab. 3). Fixing noisy data gives better improvement than removing them. However, removing noisy data could be more cost-effective in practice because it does not require human labor.

Fixing/removing noisy data improve the training process more significantly. In Fig. 3, there is a big performance gap between models trained on noisy data and models trained on data with the noise fixed/removed. Although the gap gets smaller as models converge, models trained on noisy data always have the lowest accuracy and converge the slowest. Intuitively, noisy examples make the gradient less informative and slow down the training. As the training progresses, the model approaches the optimum and the gradient norm, and therefore, the gradient noise becomes smaller. The effect of noise, therefore, also decreases as the model converges.

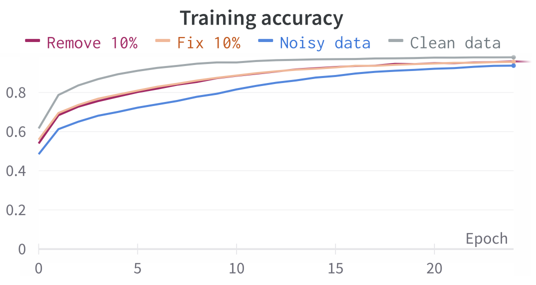


Figure 3: Training accuracy on ImageNet100 dataset.

4.4 Discussion and Future Works

Unlike rule-based or statistics-based approaches, our model-based approach is easily expandable. We can use all of the machine learning techniques to improve the quality of the supervised model and therefore, the error detection accuracy.

We outline several ways to further improve our algorithms:

Remove the need for humans: in Alg. 2, we still need humans to check the cleanliness of informative examples $\mathcal{U}$. We can automate this process by using a classifier g_{ϕ}. If the label of an example $\mathbf{z} = (\mathbf{x}, \mathbf{y}) \in \mathcal{U}$ agrees with the prediction $g_{\phi}(\mathbf{x})$ with high confidence then we consider $\mathbf{z}$ a clean example. We can further improve the diversity of the data and the quality supervised models with data augmentation.

Automatic error correction can be done easily with our model-based algorithm. We simply use the output $\hat{\mathbf{y}} = f_{\theta}(\mathbf{x})$ as the new target output for input $\mathbf{x}$. This method works for both continuous and categorical data.

5 Conclusion

In this paper, we introduce a number of algorithms for error detection on large-scale deep learning datasets. We show that influence functions can be applied to the error detection problem. We provide a theoretical and empirical explanation for the instability of gradient-based error detection algorithms. The model-based error detection algorithm is faster, more stable, and more performant than gradient-based algorithms. There is a lot more room for this model-based approach to grow. We believe that model-based error detection will be the main solution to this challenging problem.

Table 1: Details of datasets and models used in our experiments.

Dataset	**Model**	**#Classes**	**#Train samples**	**#Noisy samples**	**#Val samples**	**#Test samples**
ImageNet100	VGG16	100	31927	6385	7482	7982
Ohsumed	BERT	23	10433	2086	6367	6366
SST-5	BERT	5	8544	1708	1101	2210
TweetEval	BERT	4	3257	651	374	1421

Table 2: Error detection accuracy

	Method	*ImageNet100*			*Ohsumed*		
		Top 5%	*Top 10%*	*Top 20%*	*Top 5%*	*Top 10%*	*Top 20%*
Random noise	*EUC*	3.07 ± 1.49	3.73 ± 1.42	4.76 ± 1.62	13.28 ± 1.30	14.14 ± 0.82	15.22 ± 0.26
	COS	26.93 ± 2.51	25.49 ± 1.67	23.62 ± 1.01	28.65 ± 1.38	27.13 ± 1.30	25.49 ± 1.28
	DOT	52.52 ± 7.12	44.75 ± 5.49	37.19 ± 3.38	18.33 ± 2.64	17.76 ± 1.77	18.12 ± 1.50
	IF	20.96 ± 0.84	21.54 ± 0.98	21.01 ± 0.64	55.62 ± 8.53	49.88 ± 6.67	41.76 ± 4.00
	GD	21.34 ± 1.07	20.67 ± 0.68	20.90 ± 0.66	54.61 ± 9.30	48.97 ± 7.43	41.06 ± 4.89
	GC	20.60 ± 1.00	20.57 ± 0.91	21.04 ± 0.66	53.55 ± 5.18	48.87 ± 4.46	42.08 ± 2.83
	TracIn	21.62 ± 0.79	21.22 ± 0.23	20.57 ± 0.69	41.80 ± 23.29	36.32 ± 17.98	32.59 ± 11.99
	Model	**96.01 ± 1.34**	**92.99 ± 1.08**	**81.41 ± 1.08**	**63.25 ± 1.44**	**60.04 ± 0.72**	**53.93 ± 1.02**
Structured noise	*EUC*	15.88 ± 1.52	15.69 + 1.40	16.30 ± 1.27	17.37 ± 2.42	17.54 ± 0.75	18.10 ± 0.73
	COS	*21.95 ± 0.50*	21.67 ± 0.71	21.46 ± 0.24	21.45 ± 0.92	21.36 ± 1.34	20.44 ± 0.42
	DOT	*29.09 ± 2.78*	27.36 ± 1.69	25.52 ± 1.08	21.07 ± 1.60	21.89 ± 1.00	21.15 ± 0.65
	IF	20.65 ± 1.77	20.99 ± 0.86	21.44 ± 0.48	58.35 ± 12.83	49.57 ± 8.15	39.59 ± 5.40
	GD	*20.04 ± 1.45*	20.11 ± 0.41	20.39 ± 0.42	59.12 ± 10.60	50.48 ± 3.59	41.45 ± 3.40
	GC	*20.04 ± 1.09*	20.25 ± 0.36	20.75 ± 0.30	59.89 ± 30.64	48.66 ± 19.90	44.24 ± 11.70
	TracIn	*21.21 ± 1.46*	20.02 ± 0.30	20.81 ± 0.08	**69.48 ± 4.19**	**62.66 ± 7.46**	49.56 ± 8.46
	Model	***96.42 ± 1.15***	**93.63 ± 1.11**	**82.12 ± 1.17**	56.29 ± 4.55	54.84 ± 3.69	**50.66 ± 2.41**

	Method	*SST-5*			*TweetEval*		
		Top 5%	*Top 10%*	*Top 20%*	*Top 5%*	*Top 10%*	*Top 20%*
Random noise	*EUC*	16.16 ± 6.55	15.31 ± 5.50	16.13 ± 4.48	17.13 ± 3.08	15.93 ± 2.55	15.90 ± 1.41
	COS	29.74 ± 2.25	28.43 ± 1.24	25.79 ± 1.33	35.65 ± 5.55	33.16 ± 5.24	29.72 ± 4.36
	DOT	18.50 ± 7.74	18.82 ± 4.09	18.75 ± 2.07	15.59 ± 7.28	15.69 ± 4.40	16.32 ± 3.24
	IF	29.10 ± 6.09	24.94 ± 1.68	20.29 ± 1.86	55.86 ± 17.97	45.54 ± 14.86	32.76 ± 8.06
	GD	28.87 ± 6.01	25.50 ± 2.42	21.21 ± 2.38	54.32 ± 16.06	45.16 ± 14.61	31.30 ± 8.21
	GC	25.64 ± 5.86	24.09 ± 5.12	23.71 ± 4.30	61.11 ± 6.19	52.38 ± 6.93	45.82 ± 4.11
	TracIn	29.39 ± 8.00	26.96 ± 5.23	23.39 ± 1.70	50.00 ± 4.69	44.54 ± 5.36	35.52 ± 4.92
	Model	**71.90 ± 8.17**	**63.23 ± 5.23**	**50.19 ± 1.70**	**85.49 ± 4.69**	**79.39 ± 5.36**	**67.13 ± 4.92**
Structured noise	*EUC*	11.19 ± 2.55	12.09 ± 2.05	13.67 ± 1.23	16.67 ± 10.39	16.69 ± 7.93	16.21 ± 7.12
	COS	29.68 ± 2.27	29.16 ± 2.70	27.97 ± 2.18	28.71 ± 11.18	26.00 ± 9.54	24.58 ± 8.95
	DOT	15.99 ± 4.64	16.86 ± 4.62	18.00 ± 3.75	21.76 ± 12.68	21.62 ± 11.58	22.85 ± 7.55
	IF	64.58 ± 18.13	48.92 ± 15.92	29.67 ± 10.02	69.12 ± 8.51	60.08 ± 7.94	46.39 ± 11.46
	GD	62.30 ± 20.35	49.10 ± 17.38	30.09 ± 10.13	68.67 ± 9.05	59.08 ± 8.83	44.24 ± 12.69
	GC	56.03 ± 23.97	42.68 ± 13.07	33.51 ± 2.61	61.73 ± 24.28	58.00 ± 12.90	51.88 ± 2.13
	TracIn	38.41 ± 26.20	36.10 ± 14.50	30.83 ± 10.60	72.84 ± 9.22	63.38 ± 4.81	47.73 ± 10.10
	Model	**85.54 ± 7.37**	**75.44 ± 5.53**	**59.12 ± 1.39**	**87.81 ± 3.36**	**82.69 ± 2.53**	**70.74 ± 2.32**

Table 3: Improvement in classification accuracy on test sets

Dataset	**Random noise**			**Structured noise**		
	Original	**Removed**	**Fixed**	**Original**	**Removed**	**Fixed**
ImageNet100	81.02 ± 0.45	**81.42 ± 2.50**	**81.90 ± 0.41**	78.66 ± 0.92	**80.20 ± 0.45**	**80.94 ± 0.37**
Ohsumed	44.94 ± 0.49	**45.39 ± 0.64**	**45.69 ± 0.66**	44.68 ± 0.44	**44.30 ± 1.35**	**45.48 ± 0.35**
SST-5	49.03 ± 1.83	**50.21 ± 1.03**	**51.62 ± 1.50**	50.16 ± 1.54	**50.16 ± 0.30**	**50.01 ± 0.44**
TweetEval	77.31 ± 2.47	**79.40 ± 1.45**	**78.37 ± 1.04**	76.36 ± 2.29	**76.92 ± 1.86**	**78.64 ± 0.81**

References

Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa Anke, and Leonardo Neves. 2020. TweetEval: Unified benchmark and comparative evaluation for tweet classification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1644–1650, Online. Association for Computational Linguistics.

Elnaz Barshan, Marc-Etienne Brunet, and Gintare Karolina Dziugaite. 2020. Relatif: Identifying explanatory training samples via relative influence. In *International Conference on Artificial Intelligence and Statistics*, pages 1899–1909. PMLR.

Samyadeep Basu, Phil Pope, and Soheil Feizi. 2021. Influence functions in deep learning are fragile. In *International Conference on Learning Representations*.

Guillaume Charpiat, Nicolas Girard, Loris Felardos, and Yuliya Tarabalka. 2019. Input similarity from the neural network perspective. *Advances in Neural Information Processing Systems*, 32.

Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 458–469.

Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. 2021. Are neural nets modular? inspecting functional modularity through differentiable weight masks. In *International Conference on Learning Representations*.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Kazuaki Hanawa, Sho Yokoi, Satoshi Hara, and Kentaro Inui. 2021. Evaluation of similarity-based explanations. In *International Conference on Learning Representations*.

Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*, pages 829–846.

Zhipeng Huang and Yeye He. 2018. Auto-detect: Data-driven error detection in tables. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1377–1392.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR.

Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *Proc. VLDB Endow.*, 9(12):948–959.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature*, 521(7553):436–444.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Saif Mohammad, Felipe Bravo-Marquez, Mohammad Salameh, and Svetlana Kiritchenko. 2018. Semeval-2018 task 1: Affect in tweets. In *Proceedings of the 12th international workshop on semantic evaluation*, pages 1–17.

Felix Neutatz, Mohammad Mahdavi, and Ziawasch Abedjan. 2019. Ed2: A case for active learning in error detection. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 2249–2252.

Pouya Pezeshkpour, Sarthak Jain, Byron Wallace, and Sameer Singh. 2021. An empirical comparison of instance attribution methods for NLP. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 967–975, Online. Association for Computational Linguistics.

Minh Pham, Craig A. Knoblock, Muhao Chen, Binh Vu, and Jay Pujara. 2021. Spade: A semi-supervised probabilistic approach for detecting errors in tables. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 3543–3551. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. 2020. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930.

Olga Russakovsky. 2021. Past and future of data centric ai.

Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

Burr Settles. 2009. Active learning literature survey.

Claude Elwood Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

A Datasets and Models

In this section, we describe the respective datasets and models that we use in our experiments. Tab. 1 summarizes information about these datasets and models. We use standard datasets and models across a variety of disciplines to benchmark our method against others. Our results are comparable and reliable.

A.1 Datasets

Ohsumed: The dataset includes medical abstracts from the MeSH categories of the year 1991. The specific task was to categorize the 23 cardiovascular diseases. The original dataset could be found at `http://disi.unitn.it/moschitti/corpora.htm`.

SST-5: This dataset was introduced the first time by (Socher et al., 2013). SST (Stanford Sentiment Treebank) is a corpus with fully labeled parse trees that allows for a complete analysis of the compositional effects of sentiment in language. It consists of 11855 single sentences extracted from movie reviews. We use this dataset for the sentiment analysis problem. We use the corpus with 5 labels. The labels are negative, somewhat negative, neutral, somewhat positive, and positive. The dataset could be found at `https://deepai.org/dataset/stanford-sentiment-treebank`.

TweetEval: This dataset was introduced the first time by (Barbieri et al., 2020). It consists of 7 heterogeneous tasks in Twitter, all framed as multi-class tweet classification. All tasks have been unified into the same benchmark. We only use the Sentiment Analysis task on Emotion Recognition (Mohammad et al., 2018). The dataset consists of 11855 tweets with 4 labels: anger, joy, sadness, and optimism. The dataset could be found at `https://github.com/cardiffnlp/tweeteval`.

ImageNet100: The ImageNet (Deng et al., 2009) dataset is a benchmark in image classification. Since the computational complexity of gradient-based methods is very large, we only use a subset consist 100 classes for our experiments, each class contains about 400 images. The dataset could be found at `https://image-net.org/`.

A.2 Models

BERT (Devlin et al., 2019) stands for Bidirectional Encoder Representations from Transformers, is based on Transformers. The BERT framework was pre-trained for natural language processing tasks. We use BERT for *Ohsumed, SST-5* and *TweetEval* datasets.

VGG16: This model was introduced by (Simonyan and Zisserman, 2014). It is a classical convolutional neural network architecture. The network uses small 3 x 3 filters. We use VGG16 for *ImageNet100* dataset.

B Experimental Setup

We use the same settings for all experiments. For both BERT (Devlin et al., 2019) and VGG16 (Simonyan and Zisserman, 2014), we use AdamW (Loshchilov and Hutter, 2019) with a learning rate of $\eta = 5 \times 10^{-5}$, and $\beta = (\beta_1, \beta_2) = (0.9, 0.999)$. The models are trained with a batch size of 32 for 35 epochs. For gradient and feature-based algorithms, the best models were selected based on classification performance on the validation set for the classification task (not the validation set for error detection $\mathcal{Z}^+$). The best model for model-based algorithm was selected using $\mathcal{Z}^+$, the validation set for error detection.

For TracIn (Pruthi et al., 2020), we calculate the influence score at every epoch, from the first epoch to the best epoch.

For our method, we start with $|\mathcal{Z}'| = 500$ clean examples. The training process has $T = 5$ iterations, in each iteration, the model trains for $u = 7$ epochs. This training process is continuous, i.e. the model from the previous iteration is continuously trained in the current iteration. After each epoch, we add $r = 100$ examples to the clean set $\mathcal{Z}'$.

We run experiments on seeds $0, 5, 8, 10$ and aggregate the results by taking the mean and variance of these 4 seeds.

C Performance of gradient-based algorithms when trained on small clean dataset

In this section, we describe our experiment with gradient-based methods when the model is trained on the small clean dataset $\mathcal{Z}'$. We compare this result with the result of training the model on the entire dataset. Our experimental results are presented in the Tab. 4. Our experiments show that gradient-based algorithms perform better when the model is trained on the large noisy dataset. So, throughout our paper, we use the large noisy dataset to train the model and take the gradient of the final layer to evaluate the influence score.

Table 4: Gradient-based on small clean data

Dataset	**Method**	**Train with full data**			**Train with small clean data**		
		Top 5%	**Top 10%**	**Top 20%**	**Top 5%**	**Top 10%**	**Top 20%**
Ohsumed	**IF**	48.75	42.38	36.43	39.54	35.19	33.37
	GD	46.83	40.27	34.37	29.56	29.15	29.67
	GC	57.20	51.39	43.38	45.49	40.46	37.34
	TracIn	57.01	48.99	39.84	26.87	23.78	23.11
SST-5	**IF**	29.74	26.81	22.60	18.50	21.19	25.41
	GD	29.51	26.70	22.48	16.16	18.38	22.37
	GC	25.29	24.24	24.30	21.78	13.19	25.29
	TracIn	39.11	36.30	27.52	25.76	25.76	24.77
TweetEval	**IF**	40.12	32.31	25.04	23.46	22.46	16.44
	GD	41.36	32.31	21.35	22.84	20.62	14.44
	GC	58.02	51.38	42.55	63.58	64.00	54.53
	TracIn	35.19	29.54	22.89	22.22	17.54	13.67
ImageNet100	**IF**	21.80	21.18	20.91	20.05	20.52	20.45
	GD	20.74	20.80	20.75	21.05	20.18	20.00
	GC	20.80	20.43	20.70	20.55	20.52	20.45
	TracIn	21.30	21.05	19.80	20.86	19.99	20.99

Table 5: Classification accuracy of models trained on small clean datasets in Alg. 2. The classification accuracy is measure on test datasets.

Dataset	**Acc**	**F1**
Ohsumed	37.73 ± 0.54	36.80 ± 0.29
SST-5	54.93 ± 1.88	54.29 ± 1.63
TweetEval	85.46 ± 0.98	85.38 ± 1.14
ImageNet100	63.15 ± 2.49	62.38 ± 2.37