Infuence functions in Machine Learning tasks

Emily First, Sudeep Raja Putta, Subendhu Rongali

Abstract

Machine learning models are generally complex and it is difficult to interpret the learned parameters. It is thus hard to follow their training and behavior. It would be great if we could understand how the model is learning from the data and making predictions. To this effect, Koh and Liang [2017] showed how influence functions can be used to calculate the influence of a given training point on test prediction of a binary classification model, and reported some applications. In this work, we extend the influence functions framework to cover more Machine Learning tasks, so that they can be used more widely in this field to understand and improve training and performance. We investigate how to calculate the influence of populations of training samples, and extend the framework to regression and multi-class problems. We also look at how we can use influence functions to train a given model better.

1 Introduction

There have been huge advances in the field of Machine Learning in recent years, particularly with the resurgence of Deep Learning. The performance scores for lots of traditional problems like image recognition and NLP have gone up. However, these models are very hard to interpret intuitively, making them akin to black boxes. Lots of other Machine Learning models also have this problem of interpretability. This limits their use in high stakes applications like health-care, where they would otherwise be immensely helpful. It would be great if we could get some insights into what the model learns from the data - for example, which training points are used in making a certain prediction, and how much they influence the prediction. This kind of interpretability would also help researchers understand their models better and potentially improve their performance.

In robust statistics, influence functions Cook and Weisberg [1982] have been traditionally used to measure the influence of a training point on the model parameters. The influence of a point quantifies how much the model parameters change as a training point is up-weighted by an infinitesimal amount. However, influence functions have not found many applications in Machine Learning. Recently, Koh and Liang [2017] used influence functions to understand the influence of the training data on predictions of unseen test points. They looked at calculating the influence efficiently for binary classification problems. They also looked at some applications of this, such as training set attacks and efficient selection of crowd sourced training chunks for manual inspection.

In this project, we look at extending their work to help in other fundamental Machine Learning tasks. We first look at the mathematical framework required to calculate the influence of populations of training points. Koh and Liang [2017] propose this as a new direction. This is an interesting problem to solve since it has good potential applications. It would help us understand how different training populations affect our predictions and also give us a concrete estimate of how our predictions would change based on the addition of new chunks of training data. Our work finds a very simple expression for this based on the individual influences calculated in Koh and Liang [2017]. We analyze the influence scores of a given population of training samples by checking how the prediction scores of the model are affected. We choose the MNIST dataset for this task since the image classification task here is quite intuitive and we can analyze influences well.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

We also investigate how we can use the work in Koh and Liang [2017] to build better prediction models by removing bad training points. Most data-sets have some amount of noise and we believe we can improve a model by eliminating the noisy training examples. The idea here is to calculate the influence scores of each of the training points on an unseen representative test sample and remove the points that consistently contribute negative influence scores for the model.

Finally, we extend the work in Koh and Liang [2017] to analyze and study regression and multi-class problems. We modify the computation of the influence scores based on the new loss functions and try to analyze the results on our data-sets. We choose the single dimension regression data-set provided in assignment 1 since it is useful for our visualizations. We use MNIST again for the multi-class problem.

The results of our work show great promise for influence functions in the field of Machine Learning. We have shown that they work well in lots of areas in Machine Learning and can be used to understand and improve performance.

2 Related Work

The primary motivation to study influence functions in the context of Machine Learning is to use them for interpreting the predictions made by ML systems. However, models such as deep neural networks are complicated and are used almost like "black boxes". There has been some work on interpreting the predictions of deep neural networks. For example, Ribeiro et al. [2016] locally fit a simpler model around the test point. Simonyan et al. [2013], Adler et al. [2016] perturb the test point to see how the prediction changes. However, this has been confined to studying models with fixed parameters and not why these parameters were learned.

Koh and Liang [2017] use influence functions to trace a model's predictions all the way back to the training data, which determine the model parameters. The concept of influence functions was first introduced in statistics in the 70s and 80s by Cook and others (Cook and Weisberg [1982, 1980], Cook [1977, 1986]). It is a classic technique from robust statistics that tells us how the model parameters change as we upweight a training point by an infinitesimal amount. In Machine Learning however, influence functions have seldom been used. Wojnowicz et al. [2016] use matrix sketching to approximate Cook's distance in generalized linear models. Cook's distance (Cook [1977]) is closely related to influence functions in the case of linear regression. Debruyne et al. [2008] use influence functions for model selection for kernel based regression. They use higher order influence functions to approximate leave one out error. Liu et al. [2014] use Bouligand influence functions to quickly estimate cross validation error in kernel methods. Bouligand influence functions introduced by Robinson [1991] are defined over continuous distributions, where as we only consider empirical distributions that are discrete. Christmann and Steinwart [2004] use influence functions to study the robustness properties of Machine Learning models that are learned by convex risk minimization.

Recently Koh and Liang [2017] used influence functions to understand the effect of training points on the test data. They also showed several applications of influence functions including understanding model behavior, debugging models, detecting dataset errors, and training set attacks. However, they only investigate the influence of a single training point on test data. In our work, we investigate population effects, i.e, the influence of a population of training points on test data. We also use influence functions to study multi-class classification and regression problems, whereas Koh and Liang [2017] only study binary classification. Additionally, we also show how we can use influence functions to remove outliers and improve model accuracy.

3 Methodology

Consider a Machine Learning problem from some input space $\mathcal{X}$ to an output space $\mathcal{Y}$. We are given training points $z_1, z_2, ..z_n$, where $z_i = (x_i, y_i)$. Let $L(z, \theta)$ be the loss for the point z and parameter $\theta \in \Theta$. The empirical risk is $\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$. The empirical risk minimizer is $\hat{\theta} = \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$. We assume that the loss function is twice differentiable and strictly convex in θ.

3.1 Upweighting a training point

We re-derive the expression for the influence of upweighting a training point from Koh and Liang [2017] for the sake of completeness. Let $\hat{\theta}_{\epsilon,z}$ be the new ERM when point z is upweighted by an infinitesimal quantity ϵ, i.e, $\hat{\theta}_{\epsilon,z} = \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta) + \epsilon L(z,\theta)$. The influence of upweighting z on ERM parameter $\hat{\theta}$ is given by:

$$\mathcal{I}_{z,params} = \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}\nabla_\theta L(z,\hat{\theta})$$

Here $H_{\hat{\theta}} = \frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i,\hat{\theta})$ is the Hessian of the empirical risk with respect to θ at $\hat{\theta}$. We derive this expression here. By the definition of $\hat{\theta}_{\epsilon,z}$:

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta L(z_i,\hat{\theta}_{\epsilon,z}) + \epsilon\nabla_\theta L(z,\hat{\theta}_{\epsilon,z}) = 0\\
\implies &\left(\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i,\hat{\theta}_{\epsilon,z})\right)\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon} + \nabla_\theta L(z,\hat{\theta}_{\epsilon,z}) + \epsilon\nabla_\theta^2 L(z,\hat{\theta}_{\epsilon,z})\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon} = 0\\
\implies &\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon} = -\left(\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i,\hat{\theta}_{\epsilon,z}) + \epsilon\nabla_\theta^2 L(z,\hat{\theta}_{\epsilon,z})\right)^{-1}\nabla_\theta L(z,\hat{\theta}_{\epsilon,z})\\
\implies &\left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0} = -\left(\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i,\hat{\theta})\right)^{-1}\nabla_\theta L(z,\hat{\theta}) = -H_{\hat{\theta}}^{-1}\nabla_\theta L(z,\hat{\theta})
\end{aligned}$$

Now we apply the chain rule to derive the expression for the influence of upweighting z on the test point z_{test}.

$$\begin{aligned}
\mathcal{I}_{z,loss}(z,z_{test}) &= \left.\frac{dL(z_{test},\hat{\theta}_{\epsilon,z})}{d\epsilon}\right|_{\epsilon=0}\\
&= \nabla_\theta L(z_{test},\hat{\theta})^\top \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0}\\
&= -\nabla_\theta L(z_{test},\hat{\theta})^\top H_{\hat{\theta}}^{-1}\nabla_\theta L(z,\hat{\theta})
\end{aligned}$$

3.2 Upweighting a population of training point

Let $\hat{\theta}_{\epsilon,Z}$ be the new ERM when the points $z \in Z$ are upweighted by an infinitesimal amount ϵ, i.e, $\hat{\theta}_{\epsilon,Z} = \arg\min_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta) + \epsilon\sum_{z\in Z} L(z,\theta)$. The influence of upweighting the population Z on the parameter $\hat{\theta}$ is given by:

$$\mathcal{I}_{Z,params} = \left.\frac{d\hat{\theta}_{\epsilon,Z}}{d\epsilon}\right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}\Big(\sum_{z\in Z}\nabla_\theta L(z,\hat{\theta})\Big)$$

The derivation is similar to the case of upweighting a single point. By definition of $\hat{\theta}_{\epsilon,Z}$:

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta L(z_i,\hat{\theta}_{\epsilon,Z}) + \epsilon\sum_{z\in Z}\nabla_\theta L(z,\hat{\theta}_{\epsilon,Z}) = 0\\
\implies &\left(\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i,\hat{\theta}_{\epsilon,Z})\right)\frac{d\hat{\theta}_{\epsilon,Z}}{d\epsilon} + \sum_{z\in Z}\nabla_\theta L(z,\hat{\theta}_{\epsilon,Z}) + \epsilon\Big(\sum_{z\in Z}\nabla_\theta^2 L(z,\hat{\theta}_{\epsilon,Z})\Big)\frac{d\hat{\theta}_{\epsilon,Z}}{d\epsilon} = 0\\
\implies &\frac{d\hat{\theta}_{\epsilon,Z}}{d\epsilon} = -\left(\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i,\hat{\theta}_{\epsilon,Z}) + \epsilon\sum_{z\in Z}\nabla_\theta^2 L(z,\hat{\theta}_{\epsilon,Z})\right)^{-1}\Big(\sum_{z\in Z}\nabla_\theta L(z,\hat{\theta}_{\epsilon,Z})\Big)\\
\implies &\left.\frac{d\hat{\theta}_{\epsilon,Z}}{d\epsilon}\right|_{\epsilon=0} = -\left(\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i,\hat{\theta})\right)^{-1}\Big(\sum_{z\in Z}\nabla_\theta L(z,\hat{\theta})\Big) = -H_{\hat{\theta}}^{-1}\Big(\sum_{z\in Z}\nabla_\theta L(z,\hat{\theta})\Big)
\end{aligned}$$

We can apply the chain rule to derive the expression for the influence of upweighting population Z on a test point z_{test}

$$\begin{aligned}\mathcal{I}_{Z,loss}(z, z_{test}) &= \frac{dL(z_{test}, \hat{\theta}_{\epsilon,Z})}{d\epsilon}\Big|_{\epsilon=0} \\ &= \nabla_\theta L(z_{test}, \hat{\theta})^\top \frac{d\hat{\theta}_{\epsilon,Z}}{d\epsilon}\Big|_{\epsilon=0} \\ &= -\nabla_\theta L(z_{test}, \hat{\theta})^\top H_{\hat{\theta}}^{-1}\big(\sum_{z\in Z} \nabla_\theta L(z, \hat{\theta})\big)\end{aligned}$$

3.3 Influence in Linear regression models

In the case of linear regression, we can derive closed form expressions for Influences. The loss function is the squared loss $\frac{1}{n}\sum_{i=1}^n(\theta^\top x_i - y_i)^2$. Its derivative is $\frac{1}{n}\sum_{i=1}^n(\theta^\top x_i - y_i)x_i$ and its Hessian is $\frac{1}{n}\sum_{i=1}^n x_i x_i^\top$. We know that the empirical risk minimizing parameter is $\hat{\theta} = (\sum_{i=1}^n x_i x_t^\top)^{-1}(\sum_{i=1} x_i y_i)$.

$$\mathcal{I}_{z,params} = -H_{\hat{\theta}}^{-1}\nabla_\theta L((x,y),\hat{\theta}) = -(\frac{1}{n}\sum_{i=1}^n x_i x_i^\top)^{-1}(\hat{\theta}^\top x - y)x$$

$$\begin{aligned}\mathcal{I}_{z,loss}(z, z_{test}) &= -\nabla_\theta L((x_{test}, y_{test}),\hat{\theta})^\top H_{\hat{\theta}}^{-1}\nabla_\theta L((x,y),\hat{\theta}) \\ &= ((\hat{\theta}^\top x_{test} - y_{test})x_{test})^\top(\frac{1}{n}\sum_{i=1}^n x_i x_i^\top)^{-1}((\hat{\theta}^\top x - y)x)\end{aligned}$$

In many cases, the Hessian may be singular and the inverse will not exist. We add a regularization term to the loss function in order to make the Hessian non-singular. The loss function then becomes $\frac{1}{n}\sum_{i=1}^n(\theta^\top x_i - y_i)^2 + \frac{\lambda}{n}\theta^\top\theta$. Its derivative is $\frac{1}{n}\sum_{i=1}^n(\theta^\top x_i - y_i)x_i + \frac{\lambda}{n}\theta$ and its Hessian is $\frac{1}{n}\sum_{i=1}^n x_i x_i^\top + \frac{\lambda}{n}I$. The Hessian would always be full rank when $\lambda > 0$. We still have a closed for expression for the empirical risk minimizing parameter $\hat{\theta} = (\sum_{i=1}^n x_i x_t^\top + \lambda I)^{-1}(\sum_{i=1} x_i y_i)$.

$$\mathcal{I}_{z,params} = -H_{\hat{\theta}}^{-1}\nabla_\theta L((x,y),\hat{\theta}) = -(\frac{1}{n}\sum_{i=1}^n x_i x_i^\top + \frac{\lambda}{n}I)^{-1}((\hat{\theta}^\top x - y)x + \lambda\hat{\theta})$$

$$\begin{aligned}\mathcal{I}_{z,loss}(z, z_{test}) &= -\nabla_\theta L((x_{test}, y_{test}),\hat{\theta})^\top H_{\hat{\theta}}^{-1}\nabla_\theta L((x,y),\hat{\theta}) \\ &= ((\hat{\theta}^\top x_{test} - y_{test})x_{test} + \lambda\hat{\theta})^\top(\frac{1}{n}\sum_{i=1}^n x_i x_i^\top + \frac{\lambda}{n}I)^{-1}((\hat{\theta}^\top x - y)x + \lambda\hat{\theta})\end{aligned}$$

3.4 Efficiently Calculating Population Influence

We face similar computational challenges as addressed in Koh and Liang [2017]. We need to be able to compute $-\nabla_\theta L(z_{test}, \hat{\theta})^\top H_{\hat{\theta}}^{-1}\big(\sum_{z\in Z}\nabla_\theta L(z,\hat{\theta})\big)$ efficiently. If we have n data points and $\theta \in \mathbb{R}^p$, forming and inverting the Hessian matrix directly requires $O(np^2 + p^3)$ operations. This becomes prohibitively expensive for large models such as neural networks.

To overcome this, we use implicit Hessian-vector products (HVPs) to efficiently approximate $s_{\text{test}} = H_{\hat{\theta}}^{-1}L(z_{\text{test}}, \hat{\theta})$ and then compute $-s_{\text{test}} \cdot \big(\sum_{z\in Z}\nabla_\theta L(z,\hat{\theta})\big)$. This allows us to precompute s_{test} for each test point of interest and then efficiently compute $-s_{\text{test}} \cdot \nabla_\theta L(z,\hat{\theta})$ for each training point $z \in Z$. Then we can add the individual influences to obtain the influence of the population Z.

Like in Koh and Liang [2017], we first use the Pearlmutter [1994] trick to compute the Hessian vector product. Then we can either use Conjugate gradients(Martens [2010]) or Stochastic Estimation(Agarwal et al. [2017]) to calculate the inverse hessian vector product s_{test}. We use tensorflow to implement these approximations.

4 Data sets

We used two data-sets to support our experiments. MNIST was our primary data-set, which we modified to fit the purpose of each of the experiments. The other dataset is the regression data-set given to us in Assignment 1 Question 3.

For the influence of populations experiment, we used a reduced MNIST data-set consisting of only digits 1 and 7. We used a novel signal processing approach (described in detail in the next section) to split the 7s based on their style: 7s with a line through the middle and just plain 7s

For the multi-class experiments, we used the full MNIST data-set. The idea was to check if we could interpret the influence of data points from multiple classes on a given test sample.

The regression model experiment made use of the data-set from Assignment 1 Question 3. We wanted to be able to visualize the influence scores and the behavior of the model and the outliers, so we decided to go with a reduced 1 dimensional data-set. It is easy to see how the results would generalize over higher dimensions.

Finally, for the model improvement experiments, we decided to go with MNIST data-set again but with reduced versions of 2 classes. We wanted to run multiple experiments and get quick results without any GPU access, so we reduced the data-set to decrease the number of parameters and help speed up the training and computation of the hessian. For our experiments, we wanted to pick classes that were visually similar, so we chose 1 and 7 for one set of experiments and 4 and 9 for another.

5 Experiments

5.1 Influence of populations

We need to validate our expression for the influence of a population of samples on a given test case. We set up our experiments so that we can correlate the influence of a population to the probability score for predicting the test sample label. We want to show that the influence score is what we claim it to be: a measure of the sign and magnitude of how the population affects the probability score for the model's test label prediction. If a population has a huge positive influence score, removing those samples from training would decrease the confidence of the model in making the correct test label prediction. We should get a lower probability score for the sample prediction if we retrain the model with the partial data-set. Similarly, if a population has a huge negative score, removing the constituent samples should increase the probability score. If a population has a very low absolute influence score, we expect the probability score to change much less upon retraining without the population.

Our data-set consists of images of 1s and 7s, extracted from the full MNIST dataset. To pick a population that allows us to also make intuitive arguments, we chose the images of 7s that are all written with a dash through the middle. We describe how we do this in section 5.1.1. This sub-population of images contains 7s that are visually distinct from all the other normal 7s. We want to see if this population contributes positively or negatively in correctly predicting a test image of a normal 7 without any line. The answer is not straightforward since we don't know if a visually distinct training sample would confuse or reinforce the model. We repeat this experiment with different test images of 7s and try to explain how the influence scores help in understanding the model.

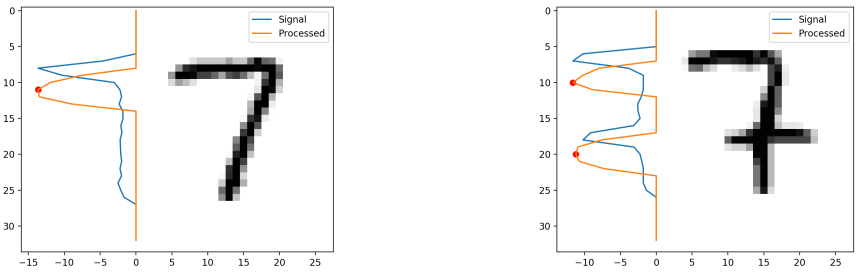
Apart from understanding positive influence scores, we also want to validate our potential negative scores. So we use Koh and Liang [2017] to pick a population of samples that have negative influence scores below a threshold for a certain test sample. We call this the bad population, with a cumulative negative influence on the test prediction. We want to show that the probability score for the correct test label increases when we remove this population and retrain the model.

5.1.1 Picking the population of dashed 7s

MNIST does not have information about the types of images. It only contains the labels. We are hence left with an unsupervised learning problem of clustering the data based on the style of the digit. This isn't a trivial problem.

The basic idea we use here is to detect how many horizontal lines are in each image. If there is just one, we know that it is a normal 7. For the dashed 7, we would get two lines. We have a 28x28

input feature set for each image and we need to find the number of lines from this. We do this by first summing the pixel intensities along the x-axis so that we get a cumulative intensity score for at each vertical coordinate. When we plot this data, we expect to see something resembling peaks where there are horizontal lines. We use techniques from signal processing to smoothen and amplify this plot and use a peak finding algorithm to detect peaks. Specifically, we project the image on its vertical axis to obtain a 1-d signal and smooth it using a Gaussian kernel. Then we use a peak finding procedure, `scipy.signal.argrelextrema` which finds relative peaks in a 1-d signal to find the number of peaks. We then extract out all the images containing two peaks as our dashed 7s population. Note that this is only an effective approximation since some badly written 7s can also throw out two peaks. We get around 650 images from this and we manually verified them to make sure that there are very few false positives.



(a) (b)

Figure 1: (a) 7 without a dash has 1 peak in its signal (b) 7 with a dash has 2 peaks in its signal. We plot the signal vertically to align it with the 7s.

5.2 Influence in regression models

For classification models, we can use the sign of the influence score as an indicator of whether the training sample has a good or bad influence on the prediction. In a regression framework, we want to understand how to interpret these scores. We would intuitively expect the influence score of a training sample to indicate whether the sample contributes to moving the regression line towards the test output or away when it is removed from the training set. We run experiments to compare these numbers.

For our experiments, we modified the standard linear regression loss function to always have a regularization term. If this term isn't present, the Hessian matrix would become singular, making it impossible for us to compute the inverse and hence the influence scores.

Apart from this, we also investigate how we can use these influence scores in outlier detection and learning robust models. With a mean squared error or similar loss functions, we always get a negative score for influence of a point from our equation.

$$L = \frac{1}{n}\sum_{i=1}^{n}(\theta^\top x_i - y_i)^2 + \frac{\lambda}{n}\theta^\top\theta$$

$$\nabla_\theta L = \frac{2}{n}\sum_{i=1}^{n}(\theta^\top x_i - y_i)x_i + \frac{2\lambda}{n}\theta$$

$$H = \nabla_\theta^2 L = \frac{2}{n}\sum_{i=1}^{n}(x_i x_i^\top) + + \frac{2\lambda}{n}I$$

Assuming the Hessian is positive semi-definite, we have influence $\mathcal{I}_{z,z} = -\nabla_\theta L(z)^\top H^{-1}\nabla_\theta L(z)$. Since the same gradient terms is used on both sides, we have a positive sign and an overall negative sign on the expression.

We expect to have slightly negative influence scores for points around the regression line, and highly negative influence scores for ones that are far away owing to their huge contribution to the loss that

affects the parameters. We can use this to infer how 'normal' a training point is in training the model and detecting outliers. For detecting outliers, we basically need to use a threshold influence score. We get this by plotting a smoothed histogram of all influence scores and choosing a value after the first apparent peak, which contains the normal points. The remaining values would all be from the outliers. We exclude the outliers found using this method from training and retrain the model.

5.3 Influence in multi-class models

Koh and Liang [2017] looked at calculating influence in binary classification problems. We run experiments that calculate influence in multi-class classification problems. We choose the full MNIST data-set for this experiment, making it a 10-class classification problem. We adapt the code-base from Koh and Liang [2017], using the same base parameters. Just as Koh and Liang [2017] used scikit-learn's Logistic Regression Pedregosa et al. [2011] for binary classification, we use the same function for multi-class classification, setting it to use cross-entropy loss and L-BFGS solver.

For each class, we manually choose one data case from the test set, selecting an image that is unambiguously in that class. We basically pick a well written digit of each class as our test sample for that class. We train our model and then calculate the influence of each of the training cases in MNIST on our selected test cases. We also calculate the Euclidean distance of each of the training cases from the test case.

We expect that training cases in the same class and that are closer in distance to the test case will have more positive influence on that test case, as they reaffirm its label. On the other hand, we expect that training cases in the same class and that are farther in distance to the test case will have more negative influence on that test case, as they confuse its label. We also expect that training cases in any of the other classes that are closer in distance to the test case will have more negative influence on that test case, as they go against its label.

5.4 Building better models using influence scores

In an attempt to build a better performing model, we hypothesize that we can remove training cases that consistently contribute negative influence scores for the model. These negative influence contributors would most likely be noisy training cases in our data-set. Thus, eliminating the noise in our data-set would improve our model. We look at the binary classification case, using two selected classes of the MNIST data-set. Specifically, we perform experiments with 1s and 7s and then 4s and 9s.

To find the training points to remove, we perform three-fold cross validation on our training data, where our folds are of the same size. Each fold is put aside as a validation set while we train our model using the other two folds. Since calculating the influence of our training cases on each of the data cases in our validation set would be too computationally expensive, we devise a method of selecting one data case from each class in the validation set for our calculations. We want this data case to be the most representative of the class in the validation set. So, for each class in the validation set, we use scikit-learn's K-means Clustering with one cluster and cluster the data cases in that class. We then find the data case closest to the center of the cluster and calculate the influence of the training cases on that data case.

Each training data case will have a set of influence scores as a result of the cross validation. We devise three heuristics for determining whether to remove the training data case. The first heuristic is to only keep training cases whose influence score set contains positive values. For the next heuristic, we only keep training cases whose influence score set sums to a positive value. The final heuristic is to only keep training cases whose influence score set contains values that are not all negative. The idea behind these heuristics is to hone in on which training cases are bad and should be removed in order to get a better model.

6 Results

6.1 Influence of populations

We used two test images to calculate our sample population influences. The influence function framework is independent of the type of Machine Learning model we choose. We went with scikit-

learn logistic regression model Pedregosa et al. [2011] to train our initial model to distinguish 1s from 7s since we get a pretty good accuracy score with very fast training. The overall model accuracy on the MNIST test data of 1s and 7s is 0.9912. We then calculate the influences with this model's parameters and loss function.

(a) (b) (c) (d)

Figure 2: (a) The first image used for the experiments. (b) A sample image from the 'bad points' population for image 1. (c) The second image used for the experiments. (d) A sample image from the 'bad points' population for image 2.

Table 1: Influence of populations on Image 1

Population	Influence	Probability score with population	Probability score without the population
Dashed 7s	595.34	0.6327	0.5768
Bad points	-583.95	0.6327	0.6651

The first test image has a very small top line for the 7, making it almost look like a 1. We find that the population of dashed 7s contributes a huge positive influence score in making the right prediction in this case. We then removed this population from the training set and retrained the model. We observed that there was a significant drop in the probability score, supporting the prediction made from the influence score that the population of dashed 7s indeed helps the model classify the given test case better.

For selecting the 'bad points' population, we used a threshold of half of the most negative influence in the training set which turned out to be -28.24. We found 15 such images from both classes with a big net negative influence score. An image from this set is shown in Figure 2. It's a 7 that almost looks like a 1. Re-training the model without these 15 images gave us a significant increase in the probability score for making the same prediction for this image. This works according to our claims about the behavior of a negatively influencing population of points.

Table 2: Influence of populations on Image 2

Population	Influence	Probability score with population	Probability score without the population
Dashed 7s	3.70	0.9959	0.9949
Bad points	-5.26	0.9959	0.9963

The second test image is a pretty good looking 7. So the population of dashed 7s has a very small positive influence score. We tried to find the reason for this by looking at some of the negatively influencing points from the 'bad points' population and we see that they contain a fair number of dashed 7s. It looks like the population on the whole contributes slightly positively in making the correct prediction but this contribution pales in comparison to what the same population contributes when the test image almost looks like a 1.

The 'bad points' population for this image consisted of 5 points below a threshold score of -0.68. This contributes a very small negative term and we find that the probability score also decreases very little. It looks like a good test image for 7 is not not greatly affected but any particular chunk of training data hugely since the model is trained well and can predict from all the remaining training examples.

Based on our results from this experiment, we have empirical proof that our expression for influence of a sample population is correct and helps us interpret the model's behavior intuitively.

6.2 Influence in regression models

We plot the training data with the input dimension on the x-axis and the output on y-axis. We want to understand what we can interpret from the sign of the influence scores on a certain test point, so we color code the influence of points by their color. We repeat this experiment for different test points in different places on the graph to see if we could find patterns and interpret what was going on. From the plots, we couldn't really interpret these values. The sign of the influence values seems to be dependent on which side of the line the training point is with respect to the test point, and also how far the test point itself is from the regression line.

The outliers however always seem to contribute a very high absolute score. We used this to build a model which ignores these outliers and builds a robust model.

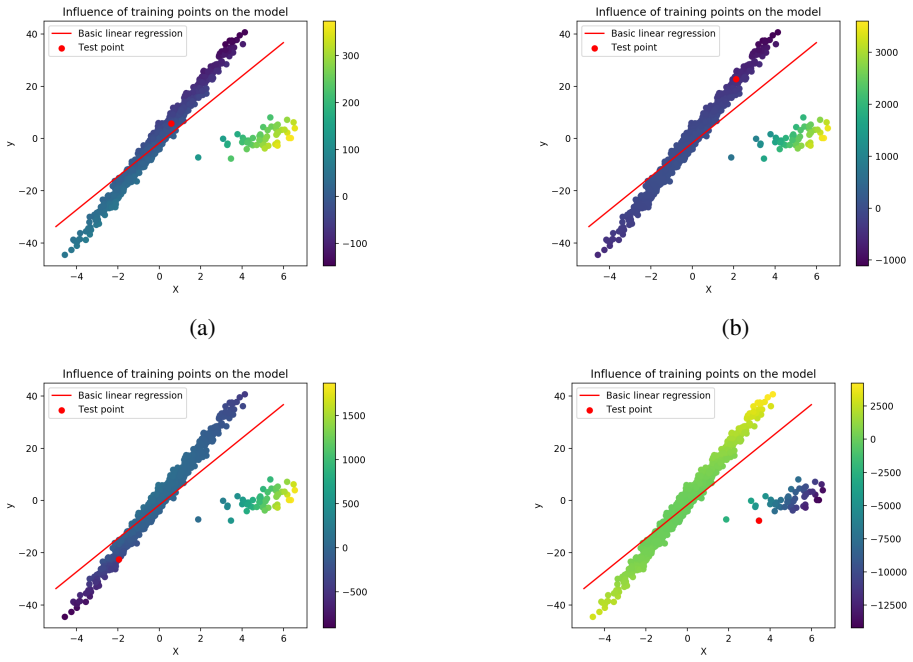
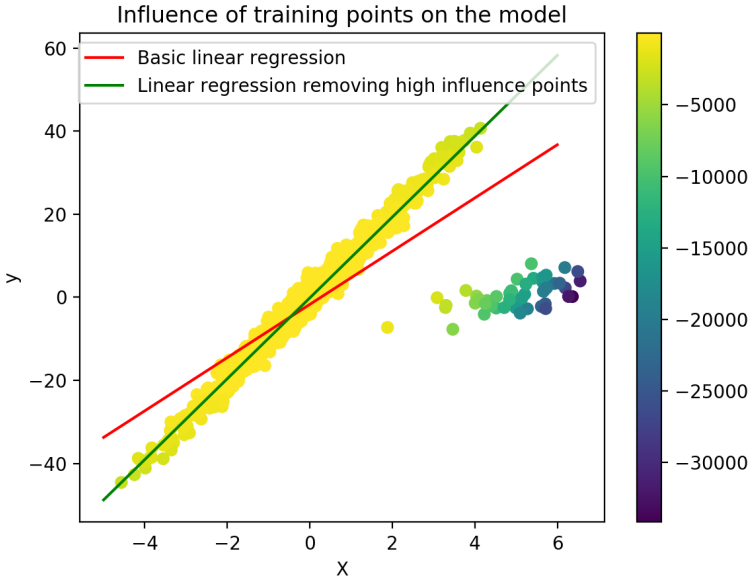


Figure 3: Model regression line and influence scores for the training points on different test points. The red dot in each graph corresponds to the test point on which the influence scores are calculated.

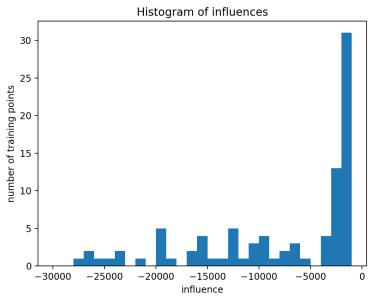
To build the robust model, we calculate the influence of each training point on itself. We explained earlier how we always get negative scores for this. We want to pick a threshold to remove all points below that threshold and retrain the model to see the new regression line. To do this, we bin the influence scores and plot a histogram of the influences. We can detect two peaks in this graph, one corresponding to the normal points and the other corresponding to the outliers. We choose a value somewhere in between these peaks as the threshold so that all the outliers can be ignored. We observed that the result line was robust to the outliers.

We observed that the influence score seemed to have a correlation with the distance of the predicted outputs from the regression line or the error in prediction. We plotted a graph with the error on x-axis and the influence score and we saw a good correlation.

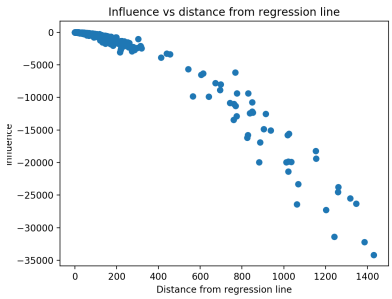
Standard robust linear regression techniques give us a similar result but we have to set the hyper-parameters optimally to make the model work well. In our approach, we need to set the threshold parameter too but it is more or less straightforward after calculating the influence scores and plotting the histogram once instead of doing a grid-search over hyper-parameters. Overall, it looked like influence functions corroborate a lot of existing techniques in regression models but they don't provide anything completely new or interpretable. All our results and plots are documented in figures 3 and 4.



(a)



(b)



(c)

Figure 4: (a) Regression lines and influence score of the training points. (b) Histogram of influences with a bin size of 1000. We pick -5000 as our threshold since the data is split at that point (b) Graph showing the relationship between our influence score and the absolute error.

6.3 Influence in multi-class models

For the experiment detailed in section 5.3 and for each test case we selected from each class, we look at the influence scores of the training cases on that test case against the Euclidean distances from the test case. As expected, the more positive influence training cases are those from the same class as the test case and fairly close in distance to the test case. It is interesting to note, though, that the training cases with the most positive influence are generally not the closest in distance to the test case. When looking at training cases that have the most negative influence on the test case, the data case is not always in the same class as the test case. It may be the instance that the training case with the most negative influence on the test case is in a different class but the image is somewhat close in distance to the test case, making it confusing for the model to classify the test case correctly.

In figure 5, we see that the most negative influence training case on a test case in class 0 is also in class 0. Our selected test case was an unambiguous 0, while the resultant negative influence training case is a 0 that does not closely resemble our test case and is even difficult to make out as a 0 at first glance.

In figure 6, we see that the most negative influence training case on a test case in class 1 is in class 7. At a glance, the resultant negative influence training case could either be a 7 or a 1. Thus, since it is

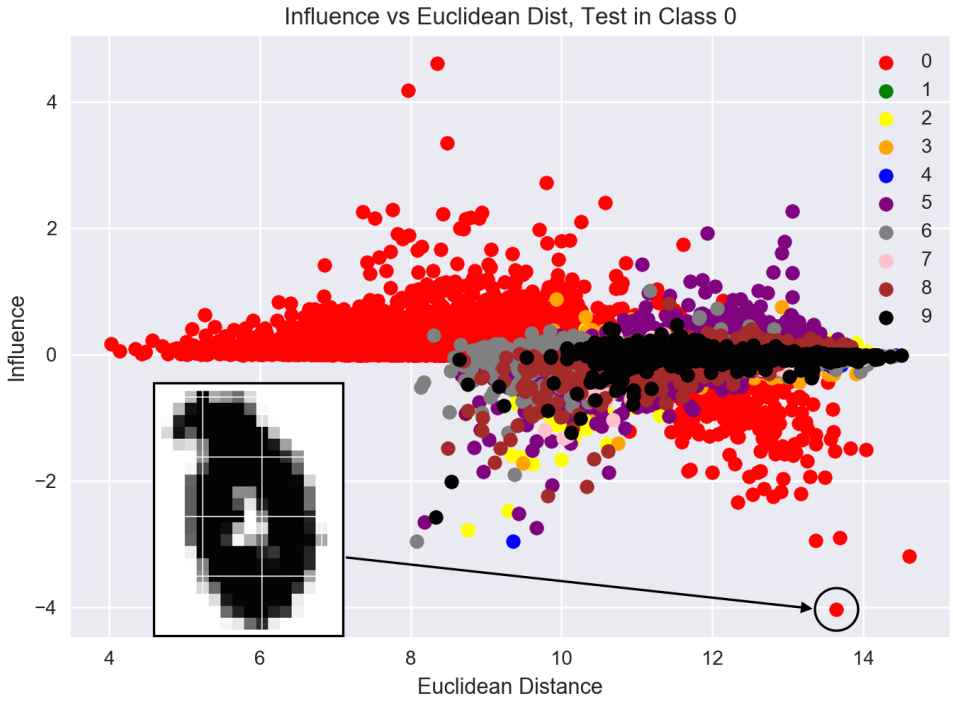


Figure 5: plot of influence scores against Euclidean distance on a test case in class 0. The most negative influence training case is also in class 0, circled and shown above. It is far in distance from the test case and does not closely resemble our 0 test case.

close in distance to our test case, it contributes negatively to the classification of the test case since its label goes against the test's label.

In figure 7, we see that the most negative influence training case on a test case in class 5 is in class 8. The negative influence training case does not look like an 8, as its top loop is not complete. The training case could conceivably be a 5, although it is a stretch. Therefore, since it is somewhat close in distance to our test case, it contributes negatively to the classification of the test case since its label goes against the test's label.

Overall, we see that the work of Koh and Liang [2017] can be extended to multi-class classification for MNIST to yield influence scores on test cases that are intuitive at a closer look.

6.4 Building better models using influence scores

For the experiments and heuristics detailed in section 5.4, the results are not very promising. We used binary classification on the MNIST data-set, with one experiment choosing only 1s and 7s and another experiment choosing only 4s and 9s.

Table 3: Building Better Model Experiment: 1 and 7

Heuristic	# Training Cases	Test Accuracy
All Training Cases	11894	0.9912
Only Positive Training Cases	1003	0.9824
Sum is Positive Training Cases	4684	0.9838
Not All Negative Training Cases	7368	0.9921

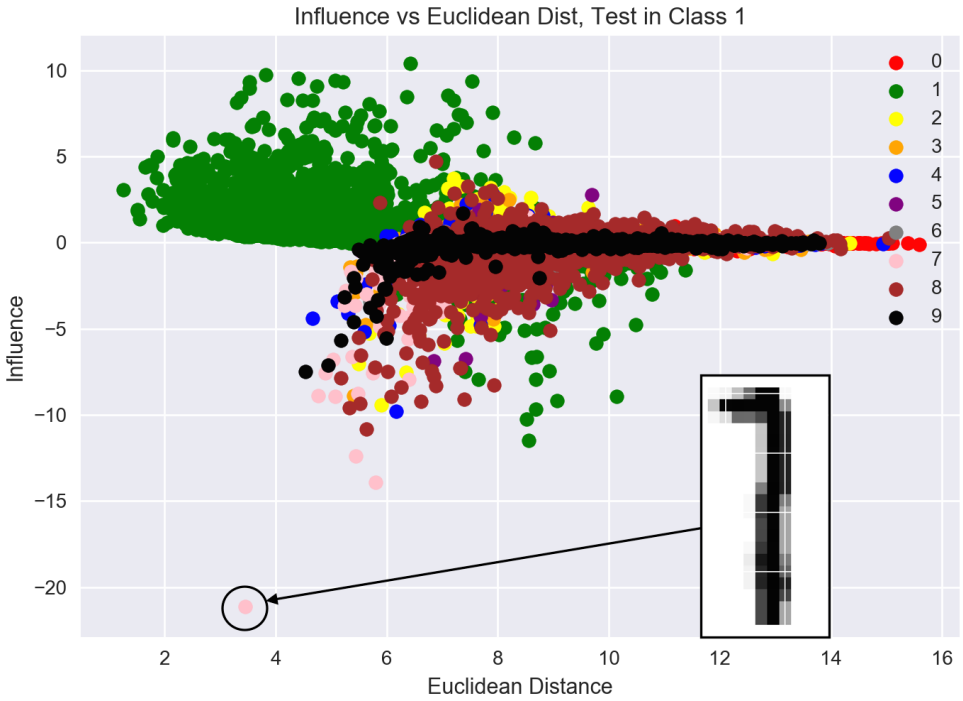


Figure 6: plot of influence scores against Euclidean distance on a test case in class 1. The most negative influence training case is in class 7, circled and shown above. It is close in distance from the test case and closely resembles our 1 test case, as this 7 looks like a 1.

When using only 1s and 7s, we found that that removing training cases based on our heuristics only yielded better accuracy on the MNIST test set when we kept training cases whose influence score set contained values that were not all negative, as detailed in table 3. The test accuracy when we use all 11894 training cases is 0.9912 as opposed to a test accuracy of 0.9921 when using 7368 training cases that are not all negative in their score sets. This is a significant result since we manage to filter out 4526 training cases, or roughly 38 percent of the training set, and achieve better test accuracy than if we had not done any filtering. Thus, we might be doing what we set out to, which is filtering out the noise in our training set in order to achieve a better performing model.

Table 4: Building Better Model Experiment: 4 and 9

Heuristic	# Training Cases	Test Accuracy
All Training Cases	10761	0.9663
Only Positive Training Cases	1064	0.9191
Sum is Positive Training Cases	4928	0.9468
Not All Negative Training Cases	8287	0.9638

When using only 4s and 9s, we found that that removing training cases based on our heuristics did not yield better accuracy on the MNIST test set and in fact produced worse test accuracies, as detailed in table 4. These unfruitful results lead us to believe that we may want to amend our process of selecting a single test case in cross validation to using the entire cross validation set. This would require more computation power, but may be worth it since our single test case chosen probably is not very indicative of the cases we would come across in the MNIST test set.

Figure 7: plot of influence scores against Euclidean distance on a test case in class 5. The most negative influence training case is in class 8, circled and shown above. This 8 can conceivably look like a 5, although is not very close in distance to the test case.

7 Discussion and Conclusions

Koh and Liang [2017] came up with a very important method for interpretability in Machine Learning models using influence functions. Their work has shown how we can efficiently calculate these values and also analyzed the results and correctness using binary classification examples. They mentioned some future directions for their work and also expressed their idea to see influence functions being used as an elementary tool to create, learn, understand, and improve Machine Learning models.

In this work, we pick up from their future work section. We derive a closed form expression to calculate the influence of a population of samples and analyze its correctness by testing on a reduced 2-class MNIST dataset. To pick intuitive populations of samples for us to better understand our estimates, we came up with ways to split the digits based on analysis of the handwriting.

To broaden the use of influence functions in Machine Learning, we also extended the ideas in Koh and Liang [2017] to multi-class and regression problems. We modified the loss functions in regression problems to ensure the influence could always be computed, and interpreted the results. We look at calculating the influence of a point on the model and compare this to the behavior of outliers. We always get negative influence scores for the training cases on themselves in regression models. So we look at the magnitude of the influence score which gives us how badly a particular point behaves in training a model and this helps us build models that are robust to outliers.

We also test the performance of influence functions in a multi-class setting. We modify the required computations based on the new multi-class cross-entropy loss function and try to interpret the influence scores we get for the training set using the full MNIST data-set. We got very intuitive results which validates the use of influence functions here. For example, we found that the most negatively contributing training image for predicting that a given test image is 1 is a training image for 7 which almost looks like a 1.

Finally, we try to use this influence framework to investigate how to build a better prediction model. The core idea is to remove training points which contribute negatively to the mode making good predictions and re-train the model on a noise-free training data-set. We have proposed multiple heuristics to choose these good points and tested them on a reduced MNIST data-set for binary classification. The results are not very promising but we explained our ideas and future directions for this. We believe our idea of picking the good test samples to calculate the influences of the training samples is a little lacking. So we could have removed training points that might potentially contribute positively towards other unseen test samples. With more computation power, we could calculate the influence scores for a more representative population and then try weeding out the bad points.

We believe this influence function framework is greatly useful to understand model behavior in Machine Learning. It would help new-comers get a better idea of what's happening, and seasoned researchers improve their models and build more complex ones without worrying too much about "black-boxifying" them. It would also be a proponent for employing Machine Learning models in high-risk tasks like health-care and mentioned earlier. We believe our work has helped improve the reach of this framework to cover more Machine Learning tasks.

References

Philip Adler, Casey Falk, Sorelle A Friedler, Gabriel Rybeck, Carlos Scheidegger, Brandon Smith, and Suresh Venkatasubramanian. Auditing black-box models by obscuring features. *arXiv preprint arXiv:1602.07043*, 2016.

Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research*, 18:116:1–116:40, 2017. URL `http://jmlr.org/papers/v18/papers/v18/16-491.html`.

Andreas Christmann and Ingo Steinwart. On robustness properties of convex risk minimization methods for pattern recognition. *Journal of Machine Learning Research*, 5(Aug):1007–1034, 2004.

R Dennis Cook. Detection of influential observation in linear regression. *Technometrics*, 19(1):15–18, 1977.

R Dennis Cook. Assessment of local influence. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 133–169, 1986.

R Dennis Cook and Sanford Weisberg. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4):495–508, 1980.

R Dennis Cook and Sanford Weisberg. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

Michiel Debruyne, Mia Hubert, and Johan AK Suykens. Model selection in kernel based regression using the influence function. *Journal of Machine Learning Research*, 9(Oct):2377–2400, 2008.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1885–1894, 2017. URL `http://proceedings.mlr.press/v70/koh17a.html`.

Yong Liu, Shali Jiang, and Shizhong Liao. Efficient approximation of cross-validation for kernel methods using bouligand influence function. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 324–332, 2014.

James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 735–742, 2010. URL `http://www.icml2010.org/papers/458.pdf`.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

Stephen M Robinson. An implicit-function theorem for a class of nonsmooth functions. *Mathematics of operations research*, 16(2):292–309, 1991.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Mike Wojnowicz, Ben Cruz, Xuan Zhao, Brian Wallace, Matt Wolff, Jay Luan, and Caleb Crable. "influence sketching": Finding influential samples in large-scale regressions. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 3601–3612. IEEE, 2016.