CIS 7000-005: Debugging Data & Models Fall 2022

Interventions — November 2

Prof. Eric Wong

In this section we'll discuss interventions—ways in which you can adjust or edit your data or models to try to fix some of these problems.

- How can you intervene at the data level (balancing, source selection)?
- How can you intervene at the model level (editing, fine-tuning)

1 Data interventionss

First, we'll look at how you can fix problems at the data level. One simple approach recently found to be competitive with start of the art is known as *data balancing*. Data balancing consists of rebalancing or reweighting your data to fix misbalanced proportions. One of the classic use-cases here is for wrost-group performance (the same setting as group DRO from the robust learning module).

Data balancing. How do you balance the data? There are two main axes by which balancing can occur:

- What subsets do you balance? One could balance by classes, which does not require extra attribute labels. However, classes may not reflect inequalities between groups. Alternatively, one could balance by groups, but this requires group attribute information.
- How do you balance? One could perform a random subset to a fixed sizeso that all groups/classes have the same size. Alternatively, one could reweight the subsets according to their sizes so that all groups/classes have the same weight during training.

The difference between weighting classes and groups is clear—one is directly related to worst-group performance but requires more data annotations, while the other is only tangentially related but can be done for free. However, what is the difference between subsetting and reweighting?

To see this, consider the following example from Sagawa et al. (2020). Let $y \in \{-1, 1\}$ be the label and $a \in \{-1, 1\}$ is a spurious attribute. Then, generate some features $x = \begin{bmatrix} x_s \\ x_c \\ x_n \end{bmatrix} \in \mathbb{R}^{d+2}$ as follows:

- $p(x_s|y, a) = \mathcal{N}(a, \sigma_s^2) \in \mathbb{R}$, which is a spurious feature that has no effect on the true label y
- $p(x_c|y, a) = \mathcal{N}(y, \sigma_c^2) \in \mathbb{R}$, which is the core feature that is correlated with the label.

- $p(x_n|y,a) = \mathcal{N}(0, \sigma_n^2) \in \mathbb{R}^d$, which are noise features that are not correlated with anything.

Here, (y, a) determines a group—we can take $y = a$ to be the majority groups, and $y = -a$ to be the minority groups. The different variances control the amount of noise in each feature. if we take σ_c to be large and σ_s, σ_n to be small, then this encourages the use of spurious and noise features before the core feature. Overparameterized models can thus memorize individual examples with x_n, and otherwise use x_s or x_c to classify larger groups.

1. A vanilla ERM model will first rely on the spurious feature which separates the majority groups, and then memorizes the minority groups with the noise vectors. This results in poor accuracy on the minority groups.
2. Subsampling the majority groups decorrelates the spurious feature with the class label. The model thus does not prioritize the majority group, and uses the core feature as the most correlated feature first.
3. Reweighting can also have a similar effect to subsampling. However, repeated training on minority examples encourages their memorization with the noise features.

What exactly does it mean for this model to memorize a datapoint? We can decompose the weights on the noise features via the representer theorem as $w_n = \sum_i \alpha^{(i)} x_n^{(i)}$ and say that the model memorizes $x^{(i)}$ if $\alpha^{(i)}$ has large weight. Intuitively, since the noise vector is high dimensional, all noise vectors of different examples are nearly orthogonal. Thus, $w_n x_n^{(i)}$ will affect the prediction on $x^{(i)}$ but not other training or test points.

Why does a model favor the spurious feature over the core feature for majority groups? A separator that depends only on the spurious feature has a norm that scales only with the number of minority data points, since it only needs to memorize the minority groups. In contrast, a separator that depends on the core features has noisier examples, so a model that uses x_c will still need to memorize some amount of all points including some in the majority group. It can be shown that this results in a larger-norm separator (since it needs to memorize more points). Since inductive bias favors minimum-norm separators, models will use the spurious feature first.

There is a nice theorem from Sagawa et al. (2020) that proves that in the overparameterized regime (i.e. for large d), there exist parameters for this setting such that the error of a max margin classifier is lower bounded by 2/3. On the other hand, for the underparameterized regime (i.e. for $d = 0$) the error is upper bounded.

Unadversarial examples If you can design the objects that your system needs to detect, then you can use adversarial examples techniques for non-nefarious purposes. Specifically, you can run an "adversarial attack" on your object to maximize the correct label, and then modify the object accordingly.

Generative modeling One other way to intervene on your data is to pass them through a generative model. For example, one can use generative models to convert data from one group to another group, or to simply generate more examples from a minority group. There are several things to watch out for here:

1. The generator needs to be diverse, and not suffer from mode collapse. This was a big problem for GANs, but may not be as big of an issue today.
2. Generative models typically needs lots of data. This is often not available for minority groups by definition. Often you can only use this approach when you already have sufficiently large data.
3. Generative models need to be high quality to be useful in this case. Otherwise, noisy or fuzzy generative models can hurt more than they help.

But if you can use a good quality generative model, then they can data augment your existing datasets to sometimes lead to big improvements.

2 Model interventions

Up to this point, we've discussed how you can adjust the data to fix any issues. As the final topic, we now ask how one can intervene on the model to fix any issues. As an example, if we ask a language model trained in 2018 "who is the president of the United States" we would likely get an out-of-date answer. How can we update or patch models, without having to retrain the model from scratch?

Suppose you have a model f_θ. Consider an input x whose desired output is y but is currently incorrect. How can we fix it? Taking a step back, what are the properties of such as fix that you'd like to have, if we update the model parameters from θ to θ'?

1. Reliability: the updated model should reliably output the correct answer y for x
2. Locality: the updated model should not affect the rest of the model for x' that are far from x
3. Generality: the updated model should correct the output for x' that are near x

With that in mind, how can we do this?

Fine tuning One natural way to do this is to fine tune on correct examples.

1. Fine-tuning tends to overfit, resulting in failures of locality and generality.
2. You can fix locality by also training on the original training set, but this is more computationally demanding and you may not have the original data.
3. Fine-tuning is not guaranteed to fix the problem—bias can still persist even if the fine tuning distribution is "fixed"!

Example: take Celeb A, which is dominated by young women and old men. The goal is to predict age.

1. The original dataset is biased and correlated women with younger ages, and men with older ages. So errors are high on old women and young men subpopulations.

2. Idea: fine tune on a small subset of perfectly balanced data.

3. Fixed feature fine tuning is still biased. Full feature fine tuning is still biased but less so. You need enough data to completely remove the bias!

3 Fixing without training

Instead of fine tuning and fixing the model with data, we can instead try to intervene and do a minimal adjustment directly on the model weights itself. For example, consider the model repair problem from Gao & Lafferty (2020) for overparameterized problems:

1. Estimate a model $\hat{\theta} \in \mathbb{R}^p$ on n datapoints (x_i, y_i).

2. Suppose the model is corrupted by noise, i.e. $\eta = \hat{\theta} + z$. where $z_j \sim (1-\epsilon)\delta_0 + \epsilon Q_j$ (i.e. corrupted by an arbitrary distribution with probability ϵ)

3. Goal: recover the original $\hat{\theta}$

Linear models Recall that the linear regression is $\hat{\theta} = X^T(XX^T)^{-1}y$. To recover $\hat{\theta}$, Gao & Lafferty (2020) propose solving

$$u = \arg\min_u \|\eta - X^T u\|_1 \tag{1}$$

and the repaired model is $X^T u$. This is effectively doing median regression onto the rows of X. They have a Theorem that says that if $\frac{\sqrt{n/p}}{1-\epsilon}$ is sufficiently small, then this recovers the original $\hat{\theta}$ with high probability.

Neural networks To handle neural networks, they do this layer by layer. First you repair the first hidden layer, i.e. solve the linear case for the X the design matrix and η the corrupted weights, and estimate the repaired weights. Then forward propagate to the second layer, and use as X the postactivations and η the corrupted second layer weights, and so on.

3.1 Model editing

Another way is to directly change the parameters of the model to fix the error. One way to do this is with another neural network (called MEND) which takes as input the gradient of the model for the correct output,$\nabla_\theta \ell(f(x), y)$ and outputs a modified gradient. This modified gradient is then used to update the model parameters with a single step.

The MEND network is trained to output gradients with several losses corresponding to the above crieteria: one loss to ensure the edit actually works, another to ensure that unrelated outputs don't change.

$$\ell_{\text{MEND}} = -\log p(y_e|x_e;\bar{\theta}) + KL(p(\cdot|x_{loc};\bar{\theta})||p(\cdot|x_{loc};\theta)) \tag{2}$$

The first term ensures that the edit is carried out to fix (x_e, y_e) while the second term ensures that the model behavior does not change on unrelated inputs.

A different way to do model editing without a classifier is via concept vectors. Concept vectors are vectors that correspond to a particular concept. A concept vector is one that linearly separates two sets of examples (and hence defines a kind of concept that discriminates between the two). This can be done at intermediate activations.

View a linear layer as a mapping from a key k to value v. Suppose (k, v) is the old concept of snow that we want to replace with road. We want to modify the layer with parameters W to remap k to a new v^* that corresponds to road.

The solve the following optimization problem:

$$\min_{\Lambda} \sum_{i} \|v_i - f(k_i; W')\| \quad \text{s.t.} W' = \Lambda (C^{-1} d)^T \tag{3}$$

where d is the top eigenvectory of the keys k and $C = \sum_d k_d k_d^T$. is a term capturing the second order statistics of the other keys. Intuitively, this is a modification of the layer parameters to rewrite the mapping in a minimal way, where minimal preserves the information of the original keys.

4 References

Salman, Hadi, et al. "When does Bias Transfer in Transfer Learning?." arXiv preprint arXiv:2207.02842 (2022).

Gao, Chao, and John Lafferty. "Model repair: Robust recovery of over-parameterized statistical models." arXiv preprint arXiv:2005.09912 (2020).

Bau, David, et al. "Rewriting a deep generative model." European conference on computer vision. Springer, Cham, 2020.

Mitchell, Eric, et al. "Fast model editing at scale." arXiv preprint arXiv:2110.11309 (2021).

Santurkar, Shibani, et al. "Editing a classifier by rewriting its prediction rules." Advances in Neural Information Processing Systems 34 (2021): 23359-23373.