Chapter 6 - Example-Based Explanations

1 Introduction

If examples are interpretable (e.g. images, text), we can select a set of examples and use that an explanation.

2 Counterfactual Explanations

This is an explanation of the form "If X had not happened, Y would not have happened". Basically, we want to know how much the features would have to change in order for the prediction to change (e.g. if you earned two hundred extra dollars a month, you would have received the loan).

How do we generate a counterfactual (x') given the original example (x) and desired target y'? Trial and error changes of x is one option, but is inefficient. A better option is to optimize a loss function that balances the similarity to the target and the closeness of the counterfactual to the original point. We have $L(x, x', y', \lambda) = \lambda \cdot (\hat{f}(x') - y')^2 + d(x, x')$. Alternatively, we can require $|\hat{f}(x') - y'| \leq \epsilon$ for a given ϵ and then optimize $\text{argmin}_{x'} \max_{\lambda} L(x, x', y', \lambda)$. We measure the distance with Manhattan distance weighted by median absolute deviation. That is, $d(x, x') = \sum_{j=1}^{p} \frac{|x_j - x'_j|}{MAD_j}$ where $MAD_j = \text{median}_{i \in \{1 \ldots n\}}(|x_{i,j} - \text{median}_{l \in \{1 \ldots n\}}(x_{l,j})|)$. This is better than Euclidean distance because it induces sparsity and is resistant to outliers.

To find the counterfactual for a given (x, y', ϵ), we do the following. We sample a random x' and optimize L. Then, while $|\hat{f}(x') - y'| > \epsilon$, we increase λ, and optimize L. Finally, we return the counterfactual x'. We can run this process multiple times to find many counterfactuals.

The counterfactual method does not require access to the model internals or training data, works with non-machine-learning models, and is easy to implement (if you use an optimization library). One disadvantage is that you need to use a good optimizer that can handle categorical variables.

3 Adversarial Examples

An adversarial example is a training example with small perturbations (that a human cannot notice) that make the model make a false prediction. For example, an attacker might generate an adversarial example to trick a computer vision model.

Most research on adversarial examples have been done on deep neural networks for object recognition. These methods take advantage of the gradient, but it is possible to generate adversarial examples in a model agnostic way (you just need to be able to run the model and generate predictions). Basically, the adversarial examples are found by minimizing $loss(\hat{f}(x + r), l) + c \cdot |r|$ where r is the perturbation, l is the desired outcome, x is the original example, and c is a parameter to balance preference between the two terms in the expression. Box-constrained L-BFGS works well to optimize it.

Another way to find adversarial examples is to add some random noise in the direction of the gradient, That is $x' \leftarrow x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y))$. This attack works well on neural networks that learn a linear relationship between pixel and score.

The one-pixel attack uses differential evolution to generate an adversarial example by changing a single pixel. Differential evolution is a genetic algorithm where we generate random candidate solutions and generate new solutions with random mutations and mixing the best candidates.

You can also print a patch (or 3D print an object) and place it next to an object to trick a deep neural net. The expectation over transformation (EOT) method allows the adversarial attack to work even when the image is transformed (e.g. rotated)

Suppose you expose a model via a web API. Most attacks will not work because they need a gradient. However, there is a way to launch a black box attack. First, you should generate some legitimate examples and get scores for them. Then you should train a surrogate model on the labeled dataset. Then, you should generate synthetic images (via a hueristic that moves pixels to maximize model score variance). Repeat the previous steps many times. Now generate adversarial examples for the surrogate model and use them to attack the web API model.

Some ways to protect against adversarial attacks include retraining your model periodically, using an ensemble of models, and including adversarial training examples in your dataset. However, you should keep adapting and coming up with new ideas because your adversaries are developing new attacks.

4 Prototypes and Criticisms

"A prototype is a data instance that is representative of all the data. A criticism is a data instance that is not well represented by the set of prototypes". The k medoids algorithm can find prototypes.

The maximum mean discrepency (MMD) critic can find both prototypes and criticisms. We greedily find prototypes (i.e. points whose distribution is similar to data distribution) and then greedily find criticisms (i.e. points whose distribution is different than data distribution). We need a kernel function k to measure density (similarity), the MMD metric to measure how similar two distributions are, and the witness function (based on kernel) to tell how different two distributions are at a given point. The MMD is measured as follows (m is number of prototypes, n is number of data points in dataset).

$$MMD^2 = \frac{1}{m^2} \sum_{i,j=1}^{m} k(z_i, z_j) - \frac{2}{mn} \sum_{i,j=1}^{m,n} k(z_i, x_j) + \frac{1}{n^2} \sum_{i,j=1}^{n} k(x_i, x_j) \tag{1}$$

Good prototypes have lower MMD^2 values. A good kernel is the radial basis function $k(x, x') = \exp(\gamma||x - x'||^2)$, where you need to pick γ. We find prototypes by greedily picking prototypes that decrease the MMD^2 value the most. The witness function is

$$witness(x) = \frac{1}{n} \sum_{i=1}^{n} k(x, x_i) - \frac{1}{m} \sum_{j=1}^{m} k(x, z_j) \tag{2}$$

A high absolute value of the witness function indicates a good criticism. We find criticisms by greedily picking points that maximize the absolute witness value plus a regularization term that enforces diversity of criticisms.

The prototypes and criticisms on their own can help you understand your model, but you can also create a simple "nearest prototype" model, which is pretty interpretable. You should run your model on the prototypes and criticisms and manually ensure that the data looks good.

MMD critic is flexible, interpretable, and easy to implement. The problem with the approach is you need to pick the number of prototypes, number of criticisms, and kernel parameter.

5 Influential Instances

An influential training instance is a training example that changes the model a lot when it is deleted and the model is retrained.

An outlier is a point that is far away from the rest of the dataset. It might be a good criticism in the dataset. It might also be an influential instance.

One way to find influential instances is with the deletion approach. One metric is $DFBETA_i = \beta - \beta^{(-i)}$, which is simply the change in the weight vector after deleting instance i. Another metric is Cook's distance, which is $D_i = \frac{\sum_{j=1}^{n}(\hat{y}_j - \hat{y}_j^{(-i)})^2}{p \cdot MSE}$. p is the number of features and MSE is the original mean squared error. Cook's distance does not require a weight vector, unlike DFBETA, but it assumes that the mean squared error is meaningful, which is not true for classification. Another option is to use $\text{Influence}^{(-i)} = \frac{1}{n}\sum_{j=1}^{n}|\hat{y}_j - \hat{y}_j^{(-i)}|$. We can also see how removing example i affects example j with $\text{Influence}_j^{(-i)} = |\hat{y}_j - \hat{y}_j^{(-i)}|$. To use these metrics, for each point, remove it, retrain model, and compute the influence of the point. The most influential points are worth investigating. We could also train an interpretable model to predict the influence of a data point to see what exactly makes influential data points influential. We could also pick a training example and see which other point's removal influences its prediction the most.

Of course, the key problem with the deletion approach is that you have to retrain the model once for each training example. To overcome this problem, we can use influence functions, but we need to have a twice differentiable loss function in order to do this.

Suppose we upweight point z. We get new parameters

$$\hat{\theta}_{\epsilon,z} = \text{argmin}_{\theta \in \Theta}(1-\epsilon)\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z, \theta) \tag{3}$$

The influence of upweighting z is then

$$I_{up,params}(z) = \frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}]_{\epsilon=0} = -H_{\hat{\theta}}^{-1}\nabla_\theta L(z, \hat{\theta}) \tag{4}$$

We can estimate the Hessian matrix with $H_{\hat{\theta}} \approx \frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i, \theta)$ (there are tricks to efficiently compute this). We can then approximate $\hat{\theta}_{-z} \approx \hat{\theta} - \frac{1}{n}I_{up,params}(z)$ and use these new parameters. If we want to see how upweighting z impacts the prediction of z_{test}, we can do it directly instead of first computing $\hat{\theta}_{-z}$ as follows $I_{up,loss}(z, z_{test}) = -\nabla_\theta L(z_{test}, \hat{\theta})^T H_{\theta}^{-1}\nabla_\theta L(z, \hat{\theta})$.

The deletion approach and influence functions allow us to compute the influence of a given point and the influence of a given point on another given point.

We can use this to figure out what training examples influence the model most overall, what makes influential instances influential (train a surrogate interpretable model), and debug why we made a wrong prediction for a test point (see the most influential training point that changes its prediction). We can also spot check influential instances in case they are actually bad data.

Influential instances are great for debugging, model agnostic, and can even be used to generate adversarial examples. Instead of looking at change in loss, change in prediction, you can look at the most influential points that affect feature importance or other metrics.

The downsides are that the deletion method is computationally expensive and influence functions only work on twice differentiable loss functions (so you cannot use them for tree based methods). If you don't have a differentiable loss, try approximating it with a differentiable one. Another limitation is that we can only consider deleting points one at a time, rather than in groups.