CIS 7000-005: Debugging Data & Models Fall 2022

Explainabilty — September 29

Prof. Eric Wong

To find problems in our models, we need ways to inspect and debug them. How can we do this for complex models often seen as opaque? This problem is called explainability in machine learning.

- What is an ideal explainability method?
- What do explainability methods currently provide?

1 Desiderate for explainabilty

What do we want out of an explanation? There are tons of different criteria one could hope for. Here's a few:

- Faithfulness - explanations should reflect an actual change in the model
- Usability - explanation should be meaningful to the end user
- Necessity - model needs the identified explanation to do a prediction
- Sufficient - explanation covers the entire model's prediction

1.1 Interpretabile by design

Some models are "interpretable by design".

Linear models

$$y = \beta^T x + \beta_0$$

What is the claim of interpretability here? it depends on the feature and the R^2.

$$R^2 = 1 - SSE/SST$$

where $SSE = \sum_i (y^i - \hat{y}^i)^2$ and $SST = \sum_i (y^i - \bar{y})^2$. If the R^2 value is high, then the the weights of the linear model reflect the variation in the data. Otherwise, if R^2 is low, then little variance is explained by the linear. In this case, the weights of the linear model can still technically be interpreted but the interpretation is virtually meaningless.

Logistic regression Linear models are typically used for regression tasks. In the case of classification, we usually use a logistic regression model instead. Is this as interpretable as a linear regression model?

$$P(y=1) = \frac{1}{1+\exp(-(\beta^T x))}$$

How to interpret the weights now? Take a log transform of the odds:

$$\ln \frac{P(y=1)}{1-P(y=1)} = w^T x \tag{1}$$

Lets call this the log odds. What happens if we increase the unit of β_i by one?

$$\frac{Odds_{x_j+1}}{Odds_{x_j}} = \exp(\beta_j) \tag{2}$$

So one unit of change corresponds to an increase in the "log odds ratio" by β_j.

1.2 Local surrogates

LIME Suppose we are happy with linear models. We can try to fit a local surrogate linear model to explain local decision boundaries. For a specific example x, model f, and linear model class G, we can solve

$$\min_g \ell(f, g; x) + \Omega(g)$$

- Select x to explain
- Perturb x locally to get a dataset of $(x_i, f(x_i))$
- Weight each x_i according to its distance to x with w_i
- Train a g on the weighted dataset
- Interpret g

For example, we can use LASSO:

$$\min_w \|w^T X - y\| + \|w\|_1$$

The first term here fits a linear model to the dataset, while the second ℓ_1 regularizer encourages a simple solution via sparsity.

Assumptions of LIME. At a first glance, interpreting with LIME appears to be a simple task—simply interpret the linear model. What assuptions are we making here?

1. We are assuming that the linear model is a good fit for the perturbed data, i.e. tha the R^2 is high and that the linear model achieves good accuracy. In other words, that the network is locally linear in the region of perturbed inputs. Does this imply faithfulness?

2. We assume a specific sparsity threshold to interpret, i.e. the number of features with non-zero weights in the linear model. This is hopefully sparse enough to imply usability.

But how many features do we interpret, or how sparse should we make the linear model? Selecting number of features is going to affect necessity and sufficiency. Furthermore, the number of datapoints you sample can affect the linear model—different runs of LIME can then output different results. Typically the feature space is too high dimensional to be able to exhaustively cover all possible perturbations. Can you construct an example where even a perfect linear model on the perturbations deviates from the neural network?

Kernel SHAP Kernel SHAP follows a similar process to LIME:

- Select x to explain
- Perturb x locally to get a dataset of $(x_i, f(x_i))$
- Weight each x_i according to the **SHAP kernel**
- Train a g on the weighted dataset
- Interpret g

Here, the main difference is that we weight each perturbed example by the SHAP kernel instead of its distance to the original datapoint. This kernel distributes the score fairly amongst the features (in a game theoretic sense). The SHAP kernel is

$$\pi(z) = \frac{M-1}{\binom{M}{|z|}|z|(M-|z|)}$$

This turns out to have some nice properties, such as

$$f(x) = E[f(x)] + \sum_j \phi_j$$

where ϕ_j are the SHAP values.

1.3 Saliencies and counterfactuals

paragraphSaliency maps Another class of method is called saliency maps. These types of methods assign a score to various input features of the model. For an input x and a model f, a saliency method assigns a score s_i for each x_i that indicates some measure of "importance" of the feature. One common way to generate scores is via the gradients of the function:

$$s = \nabla_x f(x) \tag{3}$$

However, the notion of "importance" is ambiguous: what counts as important? One might expect that important features significantly contributed towards the prediction. Can you construct an

example where a gradient saliency map assigns features a high score (gradient) that has low importance, or a low score that has high importance?

There is a swarth of saliency map methods, using either gradient-information or other model information to generate scores. For each of these methods, what exactly is the score measuring? Are you convinced that the score is correlated with importance?

Briefly, we'll go over two other classes of local explainability methods.

Counterfactuals A fourth class of explanations are counterfactual explanations. These approaches try to explain the model's prediction by answering "what if" questions. For example, what is the smallest change that a loan applicant can make that would have resulted in a credit approval instead of rejection? The main question here is how to generate such counterfactuals. One natural approach is to solve the following optimization problem:

$$\min_{z'} \lambda(f(x') - y')^2 + d(x, x') \tag{4}$$

where d is a distance metric such as the weighted manhattan distance,

$$d(x, x') = \sum_j \frac{|x_j - x'_j|}{\sigma_j} \tag{5}$$

2 Neural representations

The previous explanations were local, in the sense that they explained the prediction of a model for a particular input. We'll next discuss some global explanations—those that attempt to explain the general decision process for a model.

2.1 Feature visualization

Feature visualizations are a way to interpret the neurons within a deep network. Intuitively, these methods search for inputs that strongly activate a specifiuc neuron. For example, for images this is:

$$\max_x h(x) \tag{6}$$

where $h(x)$ is a subnetwork of f that outputs just one neuron. One can solve for this via optimization tools like gradient ascent to craft synthetic visualizations, or search over the dataset to find realistic "exemplars" from natural examples. Typically this is done in the vision setting.

Searching over natural images to find highly activating images is attractive as natural images are directly understandable. Directly optimizing doesn't work out of the box and requires a lot of handy tricks (such as regularization, generative priors, removal of high frequency signals) to make them output something human understandable, so its best to use existing libraries. Aside: standard gradient ascent actually produces more semantically meaningful results on adversarially robust models.

- Neurons can encode many different features that may be different from the top-most activating pattern. There is some work towards generating "multi faceted" visualizations, but how many visualizations do you need to truly understand one neuron?
- Are visualizations actually reflecting real patterns in data, or are we just hallucinating? Many of these visualizations are unrecognizable.
- Even if you have a perfect feature visualization, what do you do with it? Did the neural network even react to this pattern?
- There are too many neurons to look at.

2.2 Linear probing

Another tool people use to interpret deep representations is to use a linear probe. Specifically, for a given representation $h(x)$, and an auxiliary task (x, y), we fit a linear layer $g(x) = w^T h(x)$ to predict y. If the layer has high accuracy on this task, then we argue that the neural network has learned the concepts encoded in the task (x, y). This is popular in natural language, for example for predicting parts of speech or partial dependencies. The claim is that high accuracy in predicting these properties implies that the property is encoded in the representation, and that the probe found it.

- Need to be careful to have control tasks that measure the ability of a probe to independently memorize learned outputs, regardless of the actual properties.
- Use a *control task*: outputs that are structured (learnable by probe) but random (independent of linguistic properties)

Consider a part of speech tagging task. An example of a control task would be to assign a fixed but random part of speech mapping C where $y_i = C(x_i)$ is randomly sampled. This could be made more complex by sampling a different C for different types of words. Then, it is argued th at the actual property of the representation to learn the task is given by the difference of the linear probe on the given task from the randomized control.

3 References

Notes largely sourced from `https://christophm.github.io/interpretable-ml-book/`

For lots of examples on feature visualization, take a look at `https://distill.pub/2017/feature-visualization/`

For more information on linear probes, see Jjohn Hewitt's page `https://nlp.stanford.edu/~johnhew/interpreting-probes.html`.