Analysis methods in NLP: Feature attribution

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding
Motivations

Why does your model make the predictions it makes?

1. Systematicity with regard to specific phenomena
2. Robustness
3. Unwanted biases
4. Weaknesses an adversary could exploit

https://github.com/cgpotts/cs224u/blob/master/feature_attribution.ipynb
1. Integrated gradients (Sundararajan et al. 2017)
2. Gradients
3. Saliency Maps (Simonyan et al. 2013)
4. DeepLift (Shrikumar et al. 2017)
5. Deconvolution (Zeiler and Fergus 2014)
6. LIME (Ribeiro et al. 2016)
7. Feature ablation
8. Feature permutation
9. ...
Axioms

Sensitivity
If two inputs $x$ and $x'$ differ only at dimension $i$ and lead to different predictions, then feature $f_i$ has non-zero attribution.

$$M([1, 0, 1]) = \text{positive}$$
$$M([1, 1, 1]) = \text{negative}$$

Implementation invariance
If two models $M$ and $M'$ have identical input/output behavior, then the attributions for $M$ and $M'$ are identical.

Sundararajan et al. 2017
Gradients · inputs

\[ \text{InputXGradient}_i(M, x) = \frac{\partial M(x)}{\partial x_i} \cdot x_i \]
Gradients · inputs

[1]: """For both functions, the `forward` method of `model` is used. `X` is an (m x n) tensor of attributions. Use `targets=None` for models with scalar outputs, else supply a LongTensor giving a label for each example."""

[2]:
```
import torch

def grad_x_input(model, X, targets=None):
    X.requires_grad = True
    y = model(X)
    y = y if targets is None else y[list(range(len(y))), targets]
    (grads, ) = torch.autograd.grad(y.unbind(), X)
    return grads * X
```

[3]:
```
from captum.attr import InputXGradient

def captum_grad_x_input(model, X, target):
    X.requires_grad = True
    amod = InputXGradient(model)
    return amod.attribute(X, target=target)
```
Gradients · inputs

```python
[4]: from sklearn.datasets import make_classification
    from sklearn.metrics import classification_report, accuracy_score
    from torch_shallow_neural_classifier import TorchShallowNeuralClassifier

[5]: X, y = make_classification(
        n_samples=1000, n_classes=2, n_features=4, n_informative=4, n_redundant=0)

[6]: model = TorchShallowNeuralClassifier()

[7]: _ = model.fit(X, y)

    Finished epoch 1000 of 1000; error is 0.1795504391193391

[8]: X_tensor = torch.FloatTensor(X)
    y_tensor = torch.LongTensor(y)

[9]: c = captum_grad_x_input(model, X_tensor, target=y_tensor)

[10]: p = grad_x_input(model, X_tensor, targets=y_tensor)

[11]: c.mean(axis=0)

[11]: tensor([0.1145, 0.2812, 0.5429, 0.1360], grad_fn=MeanBackward1)

[12]: p.mean(axis=0)

[12]: tensor([0.1145, 0.2812, 0.5429, 0.1360], grad_fn=MeanBackward1)

[13]: pred = model.predict(X)

[14]: cpred = captum_grad_x_input(model, X_tensor, target=torch.LongTensor(pred))

[15]: cpred.mean(axis=0)

[15]: tensor([0.1259, 0.3090, 0.5372, 0.1462], grad_fn=MeanBackward1)
```
Gradients \cdot inputs fails sensitivity

\[ M(x) = 1 - \max(0, 1 - x) \]

\[
\begin{align*}
M(0) &= 1 - \max(0, 1 - 0) = 1 - 1 = 0 \\
M(2) &= 1 - \max(0, 1 - 2) = 1 - 0 = 1
\end{align*}
\]

\[
\begin{align*}
\text{InputXGradient}(M, 0) &= \max(0, \text{sign}(1 - 0)) \cdot 0 = 1 \cdot 0 = 0 \\
\text{InputXGradient}(M, 2) &= \max(0, \text{sign}(1 - 2)) \cdot 2 = 0 \cdot 2 = 0
\end{align*}
\]

Example from Sundararajan et al. 2017
Integrated gradients: Intuition
Integrated gradients: Intuition
Integrated gradients: Intuition
Core computation

\[ IG_i(M, x, x') = (x_i - x'_i) \cdot \sum_{k=1}^{m} \frac{\partial M(x' + \frac{k}{m} \cdot (x - x'))}{\partial x_i} \cdot \frac{1}{m} \]

1. Generate \( \alpha = [1, \ldots, m] \)
2. Interpolate inputs between baseline \( x' \) and actual input \( x \)
3. Compute gradients for each interpolated input
4. Integral approximation through averaging
5. Scaling to remain in the space region as the original

Adapted from the TensorFlow integrated gradients tutorial
Sensitivity again

\[ M(x) = 1 - \max(0, 1 - x) \]

\[ M(0) = 1 - \max(0, 1 - 0) = 1 - 1 = 0 \]

\[ M(2) = 1 - \max(0, 1 - 2) = 1 - 0 = 1 \]

\[ \text{InputXGradient}(M, 0) = \max(0, \text{sign}(1 - 0)) \cdot 0 = 1 \cdot 0 = 0 \]

\[ \text{InputXGradient}(M, 2) = \max(0, \text{sign}(1 - 2)) \cdot 2 = 0 \cdot 2 = 0 \]

\[ IG_i(M, 2, 0) = (2 - 0) \cdot \sum \frac{1}{m} \approx 1 \]
Feed-forward example

```
[1]: from collections import Counter
from captum.attr import IntegratedGradients
from nltk.corpus import stopwords
from operator import itemgetter
import os
from sklearn.metrics import classification_report
import torch
from torch_shallow_neural_classifier import TorchShallowNeuralClassifier
import sst

[2]: SST_HOME = os.path.join("data", "sentiment")

[3]: stopwords = set(stopwords.words('english'))

[4]: def phi(text):
    return Counter([w for w in text.lower().split() if w not in stopwords])

[5]: def fit_mlp(X, y):
    mod = TorchShallowNeuralClassifier(early_stopping=True)
    mod.fit(X, y)
    return mod

[6]: experiment = sst.experiment(
    sst.train_reader(SST_HOME), phi, fit_mlp, sst.dev_reader(SST_HOME))

Stopping after epoch 37. Validation score did not improve by tol=1e-05 for more than 10 epochs. Final error is 0.7182262241840363

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>negative</td>
<td>0.625</td>
<td>0.671</td>
<td>0.647</td>
<td>428</td>
</tr>
<tr>
<td>neutral</td>
<td>0.246</td>
<td>0.127</td>
<td>0.167</td>
<td>229</td>
</tr>
<tr>
<td>positive</td>
<td>0.634</td>
<td>0.748</td>
<td>0.686</td>
<td>444</td>
</tr>
</tbody>
</table>
Feed-forward example

```python
[7]: classifier = experiment['model']

[8]: classifier.classes_

[8]: ['negative', 'neutral', 'positive']

[9]: X_test = experiment['assess_datasets'][0]['X']
    y_test = [classifier.classes_.index(label)
              for label in experiment['assess_datasets'][0]['y']]
    preds = [classifier.classes_.index(label)
             for label in experiment['predictions'][0]]
    fnames = experiment['train_dataset']['vectorizer'].get_feature_names()

[10]: ig = IntegratedGradients(classifier.model)

[11]: baseline = torch.zeros(1, experiment['train_dataset']['X'].shape[1])

[12]: attrs = ig.attribute(
          torch.FloatTensor(X_test), baseline, target=torch.LongTensor(preds))
```
Feed-forward example

```python
[13]: def error_analysis(gold=1, predicted=2):
    err_ind = [i for i, (g, p) in enumerate(zip(y_test, preds))
               if g == gold and p == predicted]
    attr_lookup = create_attr_lookup(attrs[err_ind])
    return attr_lookup, err_ind

def create_attr_lookup(attrs):
    mu = attrs.mean(axis=0).detach().numpy()
    return sorted(zip(fnames, mu), key=itemgetter(1), reverse=True)

[14]: attrs_lookup, err_ind = error_analysis(gold=1, predicted=2)

[15]: attrs_lookup[:: 5]

[15]: [('.', 0.06881114692146112),
     ('film', 0.048555303175068946),
     ('fun', 0.04074530858858675),
     ('solid', 0.03245438354763919),
     (',', 0.028427555063823048)]

[16]: ex_ind = err_ind[0]

[17]: experiment['assess_datasets'][0]['raw_examples'][ex_ind]

[17]: 'No one goes unindicted here, which is probably for the best.'

[18]: ex_attr_lookup = create_attr_lookup(attrs[ex_ind:ex_ind+1])

[19]: [(f, a) for f, a in ex_attr_lookup if a != 0]

[19]: [('best', 0.7126857703976734),
     (',', 0.07008059173159924),
     (',', 0.02738128326101944),
     ('one', -0.040591713271602575),
     ('goes', -0.21833576011067812),
     ('probably', -0.28605132775319597)]
BERT example

\( x_4 \) \( x_7 \) \( x_{32} \) \( x_{43} \) \( x_0 \)

\( y \)
**BERT example**

```python
[1]: import torch
torch.nn.functional as F
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from captum.attr import LayerIntegratedGradients
from captum.attr import visualization as viz

[2]: weights_name = 'cardiffnlp/twitter-roberta-base-sentiment'

[3]: tokenizer = AutoTokenizer.from_pretrained(weights_name)

[4]: model = AutoModelForSequenceClassification.from_pretrained(weights_name)

[5]: def predict_one_proba(text):
    input_ids = tokenizer.encode(
        text, add_special_tokens=True, return_tensors='pt')
    model.eval()
    with torch.no_grad():
        logits = model(input_ids)[0]
        preds = F.softmax(logits, dim=1)
    model.train()
    return preds.squeeze(0)

https://captum.ai/tutorials/Bert_SQUAD_Interpret
BERT example

[6]:
```python
def ig_encodings(text):
    pad_id = tokenizer.pad_token_id
    cls_id = tokenizer.cls_token_id
    sep_id = tokenizer.sep_token_id
    input_ids = tokenizer.encode(text, add_special_tokens=False)
    base_ids = [pad_id] * len(input_ids)
    input_ids = [cls_id] + input_ids + [sep_id]
    base_ids = [cls_id] + base_ids + [sep_id]
    return torch.LongTensor([input_ids]), torch.LongTensor([base_ids])
```

[7]:
```python
def ig_forward(inputs):
    return model(inputs).logits
```
BERT example

```python
[8]: #layer = model.roberta.encoder.layer[0]
    layer = model.roberta.embeddings
    ig = LayerIntegratedGradients(ig_forward, layer)

[9]: text = "This is illuminating!"

[10]: true_class = 2  # positive

[11]: input_ids, base_ids = ig_encodings(text)

[12]: attrs, delta = ig.attribute(
    input_ids, base_ids, target=true_class, return_convergence_delta=True)

[13]: attrs.shape

[13]: torch.Size([1, 6, 768])

[14]: scores = attrs.sum(dim=-1)
    scores = (scores - scores.mean()) / scores.norm()

[15]: scores.shape

[15]: torch.Size([1, 6])
```
BERT example

```python
[16]: pred_probs = predict_one_proba(text)
[17]:
pred_class = pred_probs.argmax()
pred_class

[17]: tensor(2)

[18]: raw_input = tokenizer.convert_ids_to_tokens(input_ids.tolist()[0])
raw_input = [x.strip("\n") for x in raw_input]

[19]: score_vis = viz.VisualizationDataRecord(
    word_attributions=scores.squeeze(0),
pred_prob=pred_probs.max(),
pred_class=pred_class,
true_class=true_class,
attr_class=None,
attr_score=attrs.sum(),
raw_input=raw_input,
convergence_score=delta)

[20]: _ = viz.visualize_text([score_vis])
```
**BERT example**

<table>
<thead>
<tr>
<th>Legend:</th>
<th>Negative</th>
<th>Neutral</th>
<th>Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Label</td>
<td>Predicted Label</td>
<td>Attribution Label</td>
<td>Attribution Score</td>
</tr>
<tr>
<td>2</td>
<td>2 (0.93)</td>
<td>None</td>
<td>1.99</td>
</tr>
</tbody>
</table>
A small challenge test

<table>
<thead>
<tr>
<th>True Label</th>
<th>Predicted Label</th>
<th>Attribution Label</th>
<th>Attribution Score</th>
<th>Word Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2 (0.82)</td>
<td>None</td>
<td>2.79</td>
<td>#s They said it would be great, and they were right. #/s</td>
</tr>
<tr>
<td>0</td>
<td>0 (0.50)</td>
<td>None</td>
<td>2.09</td>
<td>#s They said it would be great, and they were wrong. #/s</td>
</tr>
<tr>
<td>2</td>
<td>2 (0.76)</td>
<td>None</td>
<td>1.38</td>
<td>#s They were right to say it would be great. #/s</td>
</tr>
<tr>
<td>0</td>
<td>0 (0.62)</td>
<td>None</td>
<td>2.62</td>
<td>#s They were wrong to say it would be great. #/s</td>
</tr>
<tr>
<td>2</td>
<td>2 (0.77)</td>
<td>None</td>
<td>1.21</td>
<td>#s They said it would be stellar, and they were correct. #/s</td>
</tr>
<tr>
<td>0</td>
<td>1 (0.47)</td>
<td>None</td>
<td>1.24</td>
<td>#s They said it would be stellar, and they were incorrect. #/s</td>
</tr>
</tbody>
</table>

**Legend:**
- Negative
- Neutral
- Positive


