Illuminating the Black Box with LIME

Ben Simmons and Alec Chen

December 2024

Introduction

Machine learning has pervaded our public consciousness and transforming a wide range of fields ranging from healthcare to finance. This integration of neural models into society poses serious risks. While we understand the fundamental mathematics behind machine learning algorithms, the mechanisms by which these models arrive at their conclusions remain elusive due to their high-dimensional, non-linear, and deeply abstracted nature. This opacity is why models are often referred to as black boxes.

A model's incorrect prediction can have critical consequences with no accountability or explanation for failure. For example, if a model misdiagnoses the disease of a patient, it could lead to improper treatment of the patient and, in the worst case, loss of life. As such, methods to interpret and explain model decision are of utmost importance. One of these algorithms is called Local Interpretable Model-Agnostic Explanations (LIME).

In this paper we re-implement LIME, an explainable AI model which attempts to interpret the "black box" of the hidden layers of another ML model. LIME allows us to approximate a more complex model with a simpler model that has more interpretable features. More specifically, we re-implement LIME within the context of text classification, aiming to find semantic meaning behind the predictions of language models.

Methods

Data

We performed text classification on the large movie review dataset from Mass et al. The large movie review dataset compromises approximately 20,000 reviews in the training set and 5000 reviews in the development set.

LSTM

We built a Long-Short Term Memory (LSTM) model to test out LIME explanation model against. The three components in this model are an embedding layer, an LSTM unit, and a fully connected layer for end-to-end learning. Our embedding layer had a vocabulary size of 10,0000 and a dimension of 100. Our recurrent layer was a LSTM cell that had input, forget, and output gates, combining these gates to regulate the flow of information through the network. Specifically, the input gate determines how much new information to incorporate into the cell state, the forget gate controls

which parts of the previous cell to discard, and the output gate decides what information to pass forward to the next time step. These gates interact with the cell state which acts as a memory mechanism and the hidden state, which acts as the current output of the LSTM. The LSTM had an hidden size of 256 and was connected to a fully connected layer which mapped the 256-dimensional hidden state to a single output, serving as the prediction layer for our task. Our model had a 89% accuracy.

LIME Implementation

In our implementation of LIME, we opted for a simplified but more focused version of the algorithm, though this choice may come at the expense of versatility and (in some circumstances) accuracy. For example, the feature selection algorithm uses forward selection, which works by removing words to determine which words (features) are most important to the classifier. There are alternative methods of feature selection, which can be faster for high feature counts, but those methods were outside the scope of this project. We used cosine similarity to for measuring the proximity of each perturbation to the original sentence, instead of providing options to use alternative proximity algorithms that could have more utility in various cases. These choices represent one of many similar decisions to maintain a language focused model without unnecessary complexity.

Our implementation, while independently developed, shares some key similarities with the original model.[1] These similarities include:

1. The feature_selection function functions in a similar way to the original LimeBase.forward_selection method. Despite exploring alternative approaches, we found the original implementation fundamentally the most optimal for this task.

2. Minor details, such as the choice of a Gaussian distribution with a kernel width of 25, or string splitting regex patterns, are used in the same manner as in the library.

3. Several core structural elements are similar to the library, such as the underlying procedure that explain_instance calls create_linear_approximations, which then calls feature_selection. However, the vast majority of the structure is unique.

However, some similarities were coincidental, such as how our SplitString class is utilized in a similar way to the lime_text.IndexedString class: we made it completely independently, but our SplitString class ended up functioning most optimally in a consolidated class similar to IndexedString.

Algorithm Description

1. Sample various points in local area around the original sentence

   (a) Generate a "mask," where the original sentence is an array of 1s, and any 0s signify a word to be removed

   (b) Generate many of those masks with randomized missing words—these are called "perturbations"

[1] Our implementation took inspiration from the original LIME implementation library in Python (https://github.com/marcotcr/lime).

(c) Calculate the distance of each perturbed mask from the original sentence's mask

2. Apply each mask to the sentence and run these perturbed sentences through the original model, providing classifications for each perturbation.
3. Select which features (words, in this case) are most significant to the original classifier
   (a) For each word, train a linear regression model to find a correlation between that word and the label (given the perturbed masks)
   (b) Choose the word with the best correlation as the first feature
   (c) Run over each word again, this time training the regression model on the combination of that word and the chosen feature
   (d) Choose the best word as your second feature, and repeat until you have as many features as needed
4. Fit a simple interpretable model (like weighted linear regression) to the chosen features, with the weights being the proximity of each point to the original sentence
5. The coefficient for each word/feature in this linear regression model can tell you how it contributes to the original classifier.

Mathematical Details

Primary equation:

$$\arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

- $\mathcal{L}(f, g, \pi_x)$: A loss function representing how accurately g(x) approximates $f(x)$, using the distance metric $\pi_x(z)$.
- $f(x)$: Original model
- $g(x)$: Interpretable model
- $\Omega(g)$: A function representing how difficult a model $g(x)$ is to interpret
- G: A set of easily interpretable models, such as linear regression models.

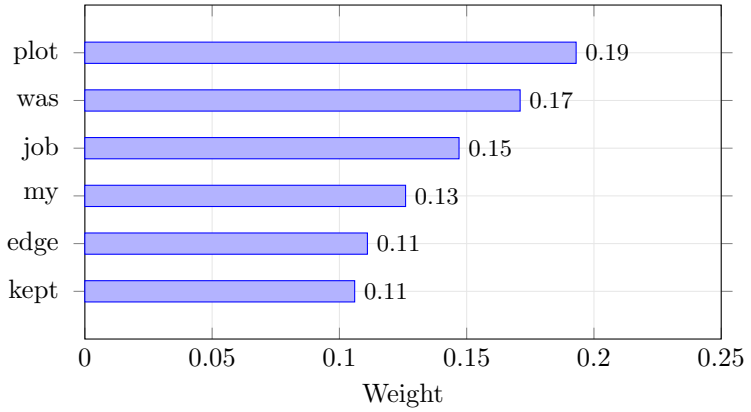
To do this we calculate the difference between our simple model and actual model prediction.
To ensure this behavior in the local neighborhood of x:

- z: Random samples using perturbations near x' based on a chosen distribution (e.g., Gaussian noise).
- z': Each z is then transformed into z', maintaining the same feature transformation as x'
- $\pi_x(z)$: proximity function assigns weights to each sampled z based on its closeness to the original input x

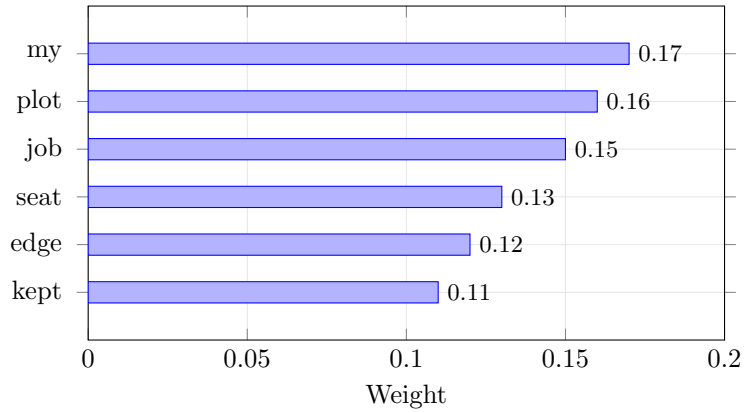
Results

LIME was was used to interpret the results from a movie review sentiment classification LSTM around the phrase "This movie was amazing! The actors did a fantastic job and the plot kept me on the edge of my seat."

Explanation for prediction using custom LIME[2]:



Explanation for prediction using original LIME[3]:



The results demonstrate that our LIME model produces very similar results to the original model. There are slight differences in the exact numbers and ordering of the features, but that is because of differences in the random number generators, not the models themselves.

Discussion

While our sampling approach produces different linear regression models than the original LIME implementation, both methods identify largely consistent sets of important words. Specifically, the

[2]absolute value of scores were taken.
[3]absolute value of scores were taken.

top six features ranked by importance remain similar between our implementation and the original LIME package

LIME demonstrated significant utility in providing explanations for complex LSTM text classification model. By perturbing and sampling the data multiple times, LIME identified the best interpretable model to approximate the LSTM's local behavior. This linear regression model achieved an $R^2 = 0.5013$ relative to the sampled local perturbations, which demonstrates an acceptable fit to the data. Using these features, LIME was able to uncover the important words that were used in the prediction of the sentiment of a sentence.

While LIME provided insights into individual predictions, we recognize certain challenges that arise when using this algorithm. The algorithm's reliance on random sampling means that minor changes in the input data could significantly alter which features are deemed most important. This instability led to inconsistent feature rankings across different runs of the algorithm on the same input.

In comparison to other explainable AI algorithms, LIME does offer advantages in lower computational complexity and ease of implementation. As a model-agnostic tool, LIME can analyze any type of model, unlike more specialized approaches like Grad-CAM (limited to convolutional neural networks) or TreeSHAP (restricted to tree-based models). However, when it comes to understanding a model's behavior across all possible inputs, other techniques like SHAP, which uses game theory principles, provider stronger capabilities for global interpretability

In conclusion, LIME is an effective explainable AI tool to extract interpretable features from more complex models with simpler models. While LIME may face some challenges with stability and consistency across multiple runs, it's computational efficiency and model agnostic nature make it a useful tool with many applications. As AI systems become increasingly integrated into our daily lives, tools like LIME play a critical role in understanding the decision making processes that underlie these technologies.

References

Gormely, Ian. "Machine Learning Interpretability: New Challenges and Approaches." Vector Institute for Artificial Intelligence, 14 Mar. 2022, vectorinstitute.ai/machine-learning-interpretability-new-challenges-and-approaches/.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).

Ribeiro, Marco Tulio, et al. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. 16 Feb. 2016, https://doi.org/10.48550/arxiv.1602.04938.

Ribeiro, Marco Tulio, et al. Local Interpretable Model-Agnostic Explanations (lime). 29 Jul. 2021, https://github.com/marcotcr/lime