LIME-Lite: Heuristic Sampling of Representative Local Points to Reduce Runtime in LIME

Ru Zhao
Department of Information Science, Cornell University
rjz46@cornell.edu

Abstract

LIME (Local Interpretable Model Agnostic Explanations) is a popular method used in Interpretable AI to explain model features. It is often used as a baseline model for many recent papers. However, the application of the model suffers from the time it takes to compute those explanations. The bottleneck of the approach requires sampling and relabeling a sizable amount of locally faithful examples in order to retrain a simpler, more interpretable classifier. In this paper, we explore the possibility of using heuristic approaches that is partially aware of the original model by sampling representative local examples in order to reduce runtime.

1 Introduction

Interpretability in NLP models has become a popular topic in recent research due to the rise of neural networks. One very popular method for explaining NLP models is LIME (Local Interpretable Model Agnostic Explanations). Decision boundaries in neural networks are arbitrary and difficult to interpret (represented by the blue/pink background seen in Figure 1), LIME's power comes from using a more interpretable model and sampling examples around an the specific instance to explain the complex model locally.

To provide context, I'll briefly describe the LIME model. For details, refer to the work by the original paper (Ribiero et al, 2016). Specifically, for an instance x, LIME sample instances around x', a more interpretable version of x such as bag-of-words. LIME draws nonzero elements of x' uniformly at random. Given a perturbed sample z' (which contains a fraction of the nonzero elements of x'), LIME recovers the sample in the original representation z and obtain f(z) by running all the samples with the original model to get labels. Then the labels are used to retrain a simpler model for the explanation. The primary intuition behind LIME is presented in Figure 1, where LIME sample instances in the vicinity of x weighted by a distance function. The distance function is associated with cosine similarity between the sample instances and the original instance.

Although LIME can be used in computer vision. For this paper, we will focus on its application in NLP. For the rest of the paper, we will describe the problem and make assumptions in the context of NLP.

Since LIME is using nonzero elements of x' samples to retrain a more interpretable model and it requires using the original model for training, it encounters the same problem with any modern machine learning models - requiring a sizable amount of training data and training time. In this case, the feature space of the nonzero elements of x' could be very large, thus randomly generating samples from that feature space to train the model should also be relatively large to represent the space. Given an instance sentence of length n, the number of samples is exponential as the number of word types in the sentence (or nonzero elements of x') increases. Table 1 details the number of possible

Word Types	**5**	**10**	**15**	**20**	**25**	**30**	**40+**
Size of feature space	32	1,024	32,765	1,048,576	33,554,432	1 billion+	1 trillion+

Table 1: Size of the feature space to be sampled with respect to the number of word types in the instance data to be explained.

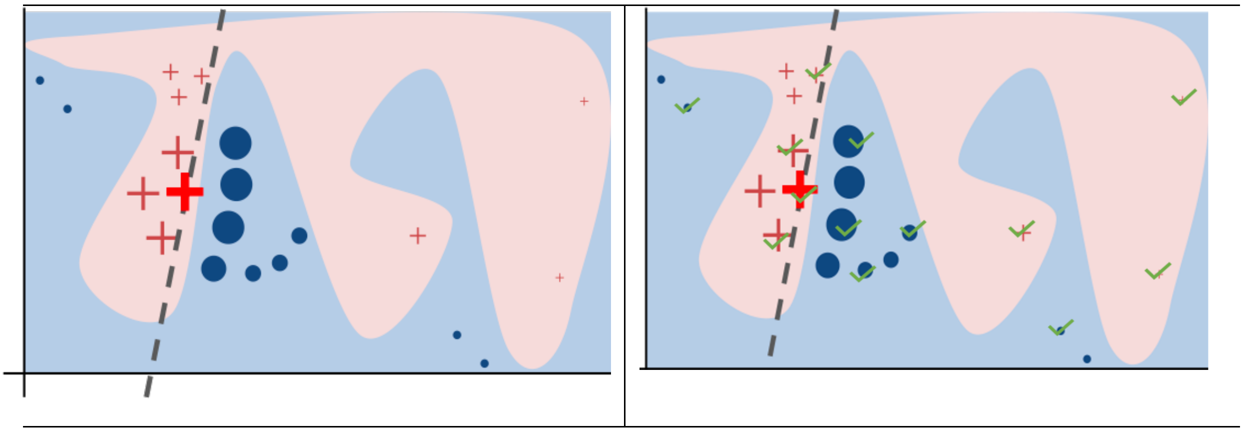

Figure 1: The toy example in the left is the original LIME's intuition. LIME samples instances, gets predictions using the original classifier, and weighs them by the proximity to the instance being explained. The dashed line is the learned explanation. The example in the right is using the heuristic approach to select samples (green checkmarks) by filtering out samples that are too closer to each other. We hope that even with less samples, we can still capture the same dashed line.

samples in the feature space with respect to the number of word types. The size of the feature space is scaled by 2^(number of nonzero elements in x'). To create an appropriate amount of samples in the feature space, it is difficult to justify that we've found a good representative subset of samples when the feature space scales to the billions with only 30+ word types. For a document size classification task, the number of possible samples would be astronomical.

Furthermore, LIME uses the original classifier to get the labels of all those examples for training the more interpretable classifier. Given inference time is long for some heavy models such as BERT (Devin et al 2018). The task to relabel an appropriate amount of samples that is representative of the feature space becomes very costly. For a sentence with more than 30 word types, there are over one billion possible combinations of features. Referring to inference speed performance in table 2, for explaining one instance with BERT, the relabeling process alone with 10,000 samples could take over 852 seconds or 14 minutes (using the cpu of Google Colab Pro).

Models	**n = 1**	**n = 10,000**
Logistic Regression	.10	3.60
GloVe-LSTM	1.96	619.27
BERT	2.70	852.94 (*estimated)

Table 2: Model inference speed in seconds per n runs for each models. BERT inference for n=10,000 is estimated based on the ratio of GloVe-LSTM for n=10,000/n=1 multiply by BERT for n=1. This is due to time constraints.

It's hard to justify that even 10,000 samples is enough to find a representative sample size for that feature space, and that would still take a long time to compute. For real world applications, people will not be running on GPUs for optimization.

For this reason, we hope to investigate heuristic approaches to find representative samples by applying a heuristic function that maximizes the representation of the sample space, by filtering from a number of redundant examples. The intuition is that if there exists samples that are clustered in the sample space, we can make sure they are a certain distance apart to filter out sizable amount of examples. Refer to Figure 2 (right) where a subset of the points in the toy example will be filtered out based on distance. Ideally we will find the samples denoted in green check marks that allow us to find the same local boundary. The task of finding an optimal subset of samples that maximizes their distance to each other can be reduced to the standard clique problem. This would make it NP-hard. Although there are other methods to select important examples, we want to examine this naïve approach as a starting point.

In LIME, consine similarity is used to weight the examples for its proximity to the instance for attributing locality. We can use the same similarity metric as a distance function for filtering out examples.

We can also utilize information from the underlying model beforehand as a filtering mechanism. This approach may jeopardize model agnosticity at the cost of speed, but a good option for those who would trade speed of retrieving the explanation for application and performance purposes. For example, for a LSTM (RNN) model that is trained on GloVe embeddings, it might be useful to use a consine distance function that is already aware of the embeddings to compute distance. We can apply a different distance functions like GloVe (Pennington 2014) that knows something about the underlying model. We then use it as a filtering process for the GloVe-LSTM model and examine whether it would perform better on that specific model in terms of performance. Similarly, information from BERT layers could be used. We'll evaluate this intuition as part of our experiments. We explore the tradeoff between employing this heuristic method and minimizing the decrease in performance of filtering out good examples.

In summary, there are many techniques that is worth exploring for reducing the number of sample instances and runtime of LIME. For this paper, we will examine a simple and naïve approach as a starting point.

2 Methods

In order to reduce runtime of LIME, we need to reduce the number of samples the original classifier has to label. If we begin with a subset of the samples by randomly sampling from the feature space, we can further filter out the subset with the approach described above. The next section, we will describe our method.

We first construct an n by n matrix where n is the number of samples. For each sample compute the distance of each point to all other points using a distance function. This distance function can be change based on appropriateness. Then we go through another round where we use it to filter the samples incrementally. The first sample we start with will be automatically selected; other samples that follow will be excluded if it's too close. Then we move on to the next sample that aren't removed and look at all of the samples that follow until we reach the end of the sample set. Below is the pseudocode for our procedure:

Pseudocode for Algorithm:

```
parameter d = distance that points can be close to each other

Construct nxn distance matrix for number of samples n.

distance_matrix = {}

for ith sample in n:
    for in sample n:
        distance_matrix[i,j]= distance_function(i, j)

selected_ samples = {}
initialize all samples to 1 in selected_samples

for ith sample in n:

    for jth sample in range(i, n) that are 1 in selected _samples:
        if distance_matrix[i,j] < d:
            deselect sample by turning it off to 0

return the list of selected samples that are 1
```

# of Word Types	5	10	15	20	25	30	40
Number of instances	611	2014	1318	1191	1085	850	649

Table 3: Distribution in the number of word types in the test set of the toxicity_pred dataset. This is the exact amount for each type.

This approach will make the function run in O(n^2) time and O(n^2) space complexity. It is unknow how many samples will be filtered, but we can adjust parameter d manually to find a desirable number of samples to be filtered per instance basis. Future work will look at methods that will allow us to select a fix set of samples.

3 Dataset

We use the toxicity dataset from Kaggle's Toxic Comment Classification Challenge to evaluate our approach. We load our data parsed with the jigsaw_toxicity_pred dataset class from HuggingFace.[1] This dataset contains comments from Wikipedia that were annotated with six different types of annotations: toxic, severely toxic, obscene, threatening, insulting, and identity hate. For our specific evaluation, we only used the toxic labels. We limit our experiments to a two-class classification task for simplicity.

Our dataset was split into training and testing set. The training set consists of 159,571 examples and the test set consists of 63,978 examples. Around 10 percent of the training examples are labelled toxic. Our pre-processing depended on the models used. We implemented three models, **Logistic Regression**, **GloVe-LSTM(RNN)** and **BERT**. For each of the models, I'll describe the preprocessing and other procedures in the descriptions of the models in the following section.

Any dataset can be used for the models and LIME, but in order to evaluate the LIME sample sizes with respect to number of word types in a sentence, we chose the toxicity dataset due to it's large number of instances and its variety in the number of word types per instance. Datasets like movie reviews are too large and generally fall in the range that is already having too large of a sample space. To examine the dataset's feasibility in identifying various sizes of word types, we partition the instances by number of word types in the test data of 63,978 test cases. Table 3 shows the number of instances that have the exact number of word types in their sentences. We can relax this to ranges to get more instances if necessary.

4 Evaluation

To evaluate what effects our approach have on identifying important representative samples, we evaluate its impact on performance with different models on different types of data instances.

We also (will) implement different distance functions with model affiliations to examine the effects of model aware heuristics on sampling.

We implemented/integrated three models with different characteristics - Logistic Regression(LR), GloVe-LSTM, and BERT. LR is very similar to the one used in the LIME paper. It will serve as a good baseline because of its inference speed compare to neural networks like GloVe-LSTM and BERT.

Below are descriptions of the models:

Logistic Regression: LR is a simple model with fast inference, we use this for comparison against current state of the art models that uses neural networks. The model is trained with default parameters from sklearn's linear classifier Logistic Regression class. When using the full dataset, the model was unable to converge. Thus we decided to use a subset of the training set. We use 10,000 of

[1] `https://huggingface.co/datasets/jigsaw_toxicity_pred`

the training samples and split the set by 80/20. We use the remaining samples as validation set for evaluation accuracy. The model was preprocessed with CountVectorizer and the resulting vectors are passed to the model.

GloVe-LSTM: The model uses GloVe embedding with an LSTM(RNN) layer. We removed stop words with NLTK stopwords corpus. We padded the data to size 200. We use the subset of the training dataset, which consists of 12,255 entries in the training set and 3,039 samples in the validation set. We train the model for 2 epochs with batch size 32 and adam optimizer. We specifically chose the 100d embeddings from the pretrained vectors of Stanford GloVe for speed.

BERT: We use an off the shelf BERT model from the HuggingFace library. The model uses the bert-base-uncased pre-trained model fine-tuned on the jigsaw_toxicity_pred dataset. The model uses the full training dataset from for training according to the description by . The model is trained for 3 epochs, with batch size 32, learning rate 2e-5, and adam optimizer.

Table 4 are the performance of the models based on validation accuracy:

Models	**Performance**
Logistic Regression	93
GloVe-LSTM	94.73
BERT	96.77

Table 4: Model Performance based on validation set.

We (will) implement three distance functions based on consine similarity of the following three methods. Below are the descriptions of the distance functions.

Distance function-1: Cosine Similarity
Compute the distance of the points using the normal consine similarity method with sckilearn.

Distance function-2: GloVe Embeddings-Cosine Similarity
Using the same word embeddings from GloVe that was used to train the GloVe-LSTM model, compute the cosine similarity.

Distance function-3: BERT Embeddings-Cosine
Using the word embeddings from BERT, compute the cosine similarity. (We can also examine the embeddings from different layers in terms of its effect on performance.)

Since the task isn't to construct explanations that is true to some annotation or faithfulness of the explanations, we can use the LIME model when it saw a larger sample as baseline. So for each instance, we can set a sample size, and apply the various models and distance functions to filter samples. We will then compute the overlap of explanations against the model as if it was able to see the larger sample. For comparison of performance, we will randomly filter samples, which is what LIME is already doing, to match the the number of samples that the distance function resulted in during the filtering process.

Table 5 and table 6 shows the explanation overlap when comparing to the original sample before

# of Word Types	**5**	**10**	**15**	**20**	**25**	**30**	**40**
Random	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*
Logistic Regression	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*
GloVe-LSTM	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*
BERT	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*	*(to be filled)*

Table 5: Explanation Overlap & Speed with respect word size and model to examine the effect of feature space on performance . Choose one of the distance functions for evaluation.

Models	Consine similarity	GloVe + consine similarity	BERT embeddings + consine similarity
Random	*(to be filled)*	*(to be filled)*	*(to be filled)*
Logistic Regression	*(to be filled)*	*(to be filled)*	*(to be filled)*
GloVe-LSTM	*(to be filled)*	*(to be filled)*	*(to be filled)*
BERT	*(to be filled)*	*(to be filled)*	*(to be filled)*

Table 6: Explanation Overlap & Speed of filtered samples compare to using the larger samples with distance d. Distance d is determined by intuition from looking at some results and how many samples are left after applying the distance filtering function).

filtering. It would also show the average speed of retrieving the explanations.

5 Results

Based on the inference speed of the various models in table 2, there is reason to believe that our approach could help in reducing the amount of time it takes to produce a good explanation. However, I ran out of time to validate our approach. Given if I was able to get results, I will compare the performance in terms speed and accuracy (annotation overlap with bigger sample) against various models, distance functions, feature space sizes, and distance parameters. It would be interesting to look at the results. I hope to continue this work in the future.

6 Discussion

In the future, we can also look for methods of heuristically generating local examples instead of filtering existing samples. It would also be interesting to look at feature embeddings of BERT of different layers to examine its correlation with performance on selecting samples.

Although our approach might be naïve, we believe this is a promising direction to investigate and I hope to continue exploring this work onward.

This project turned out to be a larger undertaking than I imagined, and I was unable to complete on time. But I hope that it still presents valuable information and methods.

Acknowledgments

Thanks to Sasha Rush and the classmates of CS6741 Spring 2021 session for the thoughtful discussions and encouragement. Specifically, I want to thank Jing Nathan Yan for helping me brainstorm. I learned a lot from everyone, including research from different areas.

References

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016, August). " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1135-1144)

S. Gonzalez, J. Landgraf and R. Miikkulainen, "Faster Training by Selecting Samples Using Embeddings," 2019 International Joint Conference on Neural Networks (IJCNN), 2019, pp. 1-7, doi: 10.1109/IJCNN.2019.8851717.

Toxic Comment Classification Challenge, Google Jigsaw, https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

Sieg, Adrien (2018) Text Similarities : Estimate the degree of similarity between two texts. https://medium.com/@adriensieg/text-similarities-da019229c894