Interpretable AI for Language Model

Sri Vikas Prathanapu,[1] **Kaushik Tummalapalli** [2] **Sai Narasimha Vayilati** [3]

[1,2,3] Department of ECE, New York University
[1] sp6904@nyu.edu, [2] kt2651@nyu.edu, [3] sv2448@nyu.edu
Repo: https://github.com/kaushik-42/Interpretable-AI-for-Language-Model

Abstract

The increasing adoption of language models, such as BERT, has led to a growing need for interpretability in AI systems. Understanding how these models make predictions and providing explanations for their decisions is crucial for building trust and ensuring accountability. The problem statement of this project is to explore and implement techniques using SHAP, LIME and Captum to interpret the predictions of BERT on a news classification task, with the goal of improving the transparency and explainability of the model.

Introduction

With the rising implementation of language models like BERT in diverse sectors ranging from healthcare, finance, to entertainment and more, the demand for interpretability in artificial intelligence (AI) systems has escalated significantly. These AI systems are often seen as "black boxes", making predictions that significantly impact decision-making processes without revealing the logic or reasoning behind their decisions.

The opaque nature of these models raises concerns about their reliability, fairness, and overall transparency. This fuels the necessity to comprehend how these models operate and derive their predictions. In order to build confidence in AI systems, ensure their accountability, and validate their predictions, explanations for their decisions are crucial. The problem statement of this project, therefore, is an exploration into the world of AI interpretability, with a specific focus on the BERT model applied to a news classification task. News classification, a form of text classification, is a hotbed for AI application, and ensuring the accuracy and fairness of such systems is of paramount importance.

To tackle this, we aim to implement and experiment with explanation techniques such as SHAP (SHapley Additive exPlanations), Lime, and Captum, (both local and global explanation methods) specifically designed to interpret the predictions made by machine learning models.

The ultimate goal of this project is to illuminate the inner workings of the BERT model when applied to news classification, demystify its decision-making process, and thereby enhance the transparency and explainability of the model. By doing so, we aim to instill greater trust in such AI systems and enable better-informed, reliable, and fair decisions in the real world.

Literature Survey

This project addresses the need for interpretability in AI systems, specifically focusing on the BERT model applied to news classification. The lack of transparency in AI systems raises concerns about reliability and fairness. To address this, the project aims to implement explanation techniques like SHAP, Lime, and Captum to interpret BERT's predictions. The main goal is to enhance the transparency and explainability of BERT in news classification. By leveraging interpretability methods, the project aims to build trust in AI systems, enable informed decision-making, and ensure fairness and accuracy in news classification tasks. These explanation techniques not only provide insights into BERT's decision-making process but also serve as valuable metrics for model evaluation and refinement. They offer explanations for the model's predictions and can be used to monitor and improve the model's performance before deployment. A specific use-case is highlighted, where interpretability methods helped identify a model's focus on "Ice" instead of "Polar Bear," leading to adjustments in model training. Interpretability models have multiple advantages, including demystifying complex AI models, instilling confidence in their predictions, and enabling informed decision-making. They also contribute to the improvement and refinement of AI models, ensuring fairness and reliability in their applications. Additionally, the abstract suggests potential economic benefits, indicating that interpretability models can assist in maximizing profits and making informed business decisions. In summary, this project focuses on interpretability in AI systems, particularly using the BERT model for news classification. By implementing techniques like SHAP, Lime, and Captum, the project aims to enhance transparency, build trust, and improve decision-making. These techniques offer both explanations and metrics for model evaluation, con-

tributing to fairness, reliability, and better outcomes in various domains.

Methodology

We will be Fine-tuning a Pre-trained BERT Model to make it relevant to our task, i.e to find out multi labels on the Reuters dataset. Coming to the Reuters dataset, we have data i.e. stored into many folders where the data is already been split and is readily available from the nltk library. The dataset has various articles along with the article headline label. The training dataset has the data start with train/id-n and the testing dataset has the data starting with test/id-n. where id-n represents the article id. We can view any particular article by mentioning the train/test along with an id number to access that respective article.

We will be creating a 'ReutersDataset' class is a custom class derived from PyTorch's 'Dataset' class. It is designed to handle the Reuters text data and transform it into a format compatible with BERT. The class uses the BERT tokenizer to convert raw text into token IDs that BERT can process. We will be having 2 data sets: one for training and one for testing. The training dataset includes all Reuters documents whose file ids start with 'training/', and the test dataset includes all documents whose file ids start with 'test/'. The categories used are simply all categories in the Reuters dataset. For each set of file-ids, a ReutersDataset object is created, which will handle tokenizing the text and converting the labels as needed. These dataset objects can then be used to train and evaluate a BERT model for text classification.

In the coming step, we will pad the sequences present in the articles using pad sequence function from the torch library, where we convert the tensor sequences to the list of integers. We will initialize 2 data loaders i.e one for the training data and the other data loader for testing data.

We will be performing the training step and also the eval step for fine-tuning the BERT model for the task at hand. Some common steps we follow: basically follow these steps: Setting the learning rate and a number of training epochs, defining the device to train on (GPU if available, else CPU), and Moving the model to the chosen device. Defining the optimizer and loss function. AdamW is often used with BERT and other transformer models because it's an adaptive optimizer that can handle sparse gradients and noisy data well. The CrossEntropyLoss is used for multi-class classification tasks. Switching the model to training mode with 'model.train()'. In the training loop, we extract the input data along with the label from the batch, encode the input data, perform any necessary padding, perform forward pass and backward pass and lastly update the model parameters. This is a typical procedure for training a model in PyTorch. There's a lot going on, but each step is crucial for ensuring that the model learns from the data. It's also quite flexible and can be modified or extended as necessary to fit different models, tasks, or training procedures.

After Fine-tuning the BERT Model for our task, we will be using various Interpretability models to explain both locally and also globally. Explanation methods are classified into Inherent Interpretable models and Post-Hoc Interpretable models. These explanations which are provided are dependent on the end-user. Post-Hoc Explanations are categorized into Local and Global Explanations. We will be looking at 3 specific Interpretation methods that are related to feature importance and summarization of the Counter Factuals. Coming to the Local Explainations, we will be using LIME and SHAP Methods to debug the model predictions and interpret that in a more comprehensive manner. As compared to the Global explanations, local explanations are just focused on feature importances, counterfactual, etc, but global explanation methods like CAPTUM provides the global picture like all the summaries of the Counter Factual or the collection of all the local explanations which can be used to judge the model on the whole instead of the features alone.

We will be comparing the results that we get from the local explanation methods and the global explanation methods.

Architecture:

For the Fine-tuned Model: In the context of your use case, you're using a fine-tuned BERT model for a multi-class text classification task. The specifics of the model, loss function, and hyperparameters can be described as follows:

1. We're using a BERT model, specifically designed and trained by Google for a wide range of NLP tasks. BERT (Bidirectional Encoder Representations from Transformers) stands out due to its bidirectional training of Transformer, a popular attention model. It is pre-trained on a large corpus of unlabelled text including the entire Wikipedia and then fine-tuned with labelled data for specific tasks. In this case, you're fine-tuning the BERT model for a multi-class text classification task.

2. The loss function you're using is the CrossEntropyLoss. This is a common choice for classification tasks. The CrossEntropyLoss function computes the cross entropy between the predictions of the model and the actual classes. The model's goal during training is to minimize this loss.

3. Hyperparameters: The learning rate that we've chosen is 2e-5 and the number of training epochs is 3. These are common choices for fine-tuning BERT models. The learning rate determines how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process. The number of epochs is the number of complete passes through the training dataset.

4. Optimizer: You're using the AdamW optimizer, a variant of the Adam optimizer with fixed weight decay. AdamW has been shown to work well with transformer models like BERT.

5. For each epoch, We will be iterating over your training data (packed in batches), applying the model to the input

Figure 1: Outline of the Fine-Tuned BERT Architecture

data, computing the loss, performing back propagation via 'loss.backward()', and then updating the model parameters with 'optimizer.step()'. The model parameters are updated in a way that minimizes the loss.

Coming to the Local Explanation Methods:

1) **LIME (Local Interpretable Model-agnostic Explanations)**: for explaining predictions of your fine-tuned BERT model for a multi-class text classification task. LIME is a method for explaining the predictions of any machine learning classifier. It works by approximating the decision boundary locally around the prediction we're interested in. In other words, it generates a simpler model (like linear regression) that approximates the predictions of the complex model in a small neighborhood around the point of interest. This simpler model is then used to interpret the prediction.

Predictor Function: The predictor function is an interface between the text data and the BERT model. It takes in a list of texts, tokenizes them, applies the BERT model, applies a softmax function to the logits to get probabilities, and then returns these probabilities. LIME uses this predictor function to query the model with perturbed versions of the original instance.

LimeTextExplainer: This is a class in the LIME library that is specifically designed for explaining predictions on text data. It generates explanations by first perturbing the input text (removing words), getting predictions for these perturbed texts using the predictor function, and then learning a linear model that approximates the behavior of the BERT model in the neighborhood of the original text.

Explaining a Prediction: When we call explain instance, LIME generates and returns an explanation for the prediction on the given instance. The num features parameter determines the number of features included in the explanation. The explanation object can then be used to visualize the explanation, for example, by highlighting words in the text according to their importance. It's important to note that the explanations generated by LIME are local and may not represent the model's behavior globally, i.e., on instances that are far from the one being explained. Also, the quality of the explanations depends on the quality of the local approximation, which may not always be perfect. The main hyperparameters:

1. number of features: This parameter determines the number of features (in this case, words) that LIME includes in the explanation. We have set thus number of features to 6. This is a major hyperparameter in the LIME explanation since it directly affects the complexity of the explanation.

2. BATCH SIZE: This is not directly a hyperparameter of LIME, but it's used in your predictor function for batching the inputs to the BERT model. It's set to 16 in your code. The BATCH SIZE can affect the speed of prediction and memory usage, but it doesn't affect the LIME explanation itself.

2) **SHAP**: the main hyperparameters and technical details are as follows:

1. max length: This is a parameter used in the 'f' function for padding/truncating the input texts. It's set to 512 in your code. This is a standard length for BERT inputs, as BERT is designed to work with sequences of this length. The choice of max length can impact the quality of your model's predictions and, consequently, the SHAP explanations.

2. Explainer initialization**: The SHAP explainer is initialized with the 'f' function and the tokenizer. In this case, no specific SHAP explainer is chosen, so the SHAP library uses a default explainer, which is a KernelExplainer for this type of model. The KernelExplainer approximates the Shapley values by sampling from a distribution over the possible inputs.

SHAP itself has a variety of hyperparameters depending on the specific explainer used. For instance, the KernelExplainer has a parameter n-samples that determines the number of times to re-evaluate the model when explaining each prediction. This parameter can significantly impact the computation time and the stability of the explanations,

3) **Captum**: Captum is a model interpretability library for PyTorch developed by Facebook. The name "Captum" is Latin for understanding or comprehension, reflecting the library's goal of providing tools for understanding and interpreting decisions made by machine learning models. 1.

Model Wrapper: To use Captum's Integrated Gradients, the forward function of the model needs to take only the input tensors to be attributed as arguments. Therefore, a wrapper around the original BERT model is created (the 'BertModelWrapper' class). This wrapper separates the embedding computation from the rest of the forward pass to satisfy the requirement. 2. Integrated Gradients (IG): IG is an

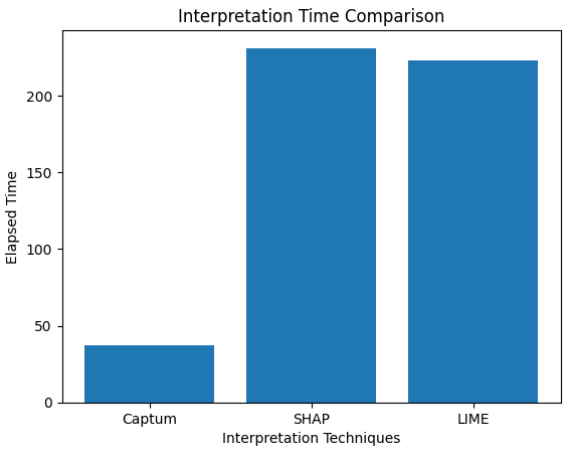

Figure 2: Interpretation Time Comparison

interpretability algorithm that assigns importance to each input feature by approximating the integral of gradients of the model's output with respect to the inputs along the path from a baseline input (usually all-zeros input) to the given input. The number of steps for the approximation is set to 500 ('n-steps=500'). This is a hyperparameter that can be adjusted, with a larger number of steps usually providing a more accurate approximation.

3. Interpretation: For each sentence to be interpreted, the 'interpret sentence' function tokenizes the sentence, computes embeddings, makes a prediction with the model, and computes attributions using IG. The attributions represent the importance of each input token to the prediction. The function also computes an approximation error ('delta'), which indicates the quality of the IG approximation. 4. Visualization: The attributions and other information are stored in 'vis-data-records-ig' for later visualization. The attributions are normalized before being stored The 'add-attributions-to-visualizer' function constructs a 'VisualizationDataRecord' object for each sentence, which includes the attributions, prediction, tokens, and delta. These records can be passed to Captum's visualization utilities to generate visual interpretations of the model's predictions.

Results

Coming to the local explanation methods: SHAP: We will get the SHAP Values from the Interpretability method, where these values suggest the importance of a given feature. SHAP takes some time to calculate the SHAP Values as it's a computationally heavy method in the local explanation space. All the outputs we get are feature words that are combined and the red and blue colour provides the shap importance.

Coming the LIME Method (also a Local Interpretability method): This method is more computationally expensive than all the Interpretability methods because it creates permutations of all the feature words and calculates the

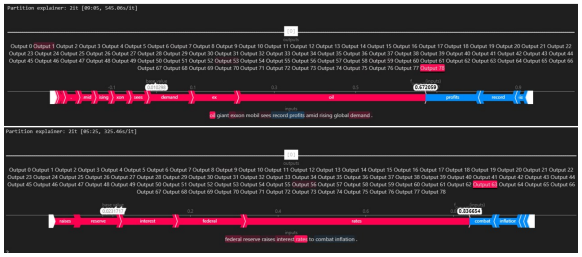

Figure 3: SHAP Result

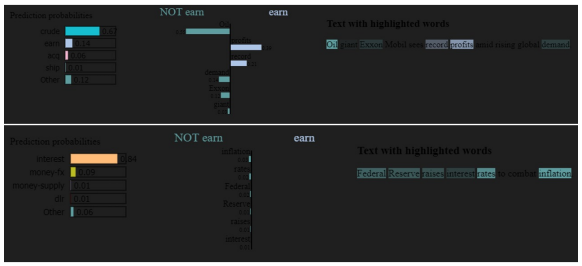

Figure 4: LIME Result

marginal contribution of that vector map which comprises the features which are included as part of the permutations. The results of the LIME model is based on the probabilities of the predicted label. As we can see, there can be multi labels as well but with different probabilities.

Coming to the Global Explanations where we will be using the CAPTUM Interpretability method, it uses various functions to highlight a particular word in the sentence which is considered a label, and If there are multiple labels, it's given as a HeatMap-like structure.

By comparing all the local and global explaination methods we tried like SHAP, LIME and CAPTUM, the prediction probabilities remain the same i.e 0.54 and the importance score is greater for LIME and SHAP methods as they are local methods which are related to the feature importance, where CAPTUM covers both the local and the global methods.

Future Improvement

To further enhance the interpretability of AI models, future work can focus on addressing the current limitations of computation time and hardware resource usage associated with Captum, SHAP, and Lime. The following areas offer potential for improvement:

1. Optimization and Efficiency: Future research should aim to optimize the computation process of Captum, SHAP, and Lime techniques, reducing the required time and computational resources. This can involve developing more efficient algorithms or exploring parallel computing approaches to expedite interpretability analysis.
2. Hardware Acceleration: Investigating the utilization of hardware accelerators like GPUs or specialized AI chips can significantly reduce processing time and resource

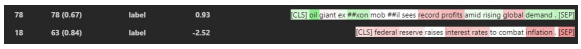

Figure 5: CAPTUM Result(Integrated Gradients)

	Technique	Prediction Probability	Importance Score	Convergence Delta
0	Integrated Gradients	0.54	0.0367	0.0068
1	SHAP	0.54	0.22	-
2	LIME	0.54	0.23	-

Figure 6: Comparing all the Interpretability methods

usage for Captum, SHAP, and Lime techniques. Customizing optimizations for these interpretability methods based on specific hardware characteristics may yield substantial performance improvements.

3. Hardware-aware Interpretability: Designing interpretability techniques that consider hardware constraints and optimize resource utilization can be advantageous. Developing adaptive methods that dynamically adjust the computation process based on available resources or building interpretable models specifically tailored to resource-constrained environments can help overcome limitations.
4. Hybrid Approaches: Exploring hybrid approaches that combine the strengths of different interpretability techniques, such as a combination of Captum, SHAP, and Lime, can provide a comprehensive and efficient interpretability solution. Leveraging the complementary nature of these methods may achieve a better trade-off between computational requirements and interpretability accuracy.

In conclusion, future work in the field of interpretability should concentrate on optimizing computation, leveraging hardware acceleration, developing hardware-aware interpretability, and investigating hybrid approaches. By addressing these aspects, researchers can pave the way for more efficient and scalable interpretability solutions applicable in real-world scenarios with limited computational resources.

Conclusion

The adoption of complex machine learning models, especially language models like BERT, has necessitated the development of robust interpretation tools for these models. Understanding the decision-making process of these models is crucial not only for improving the model's performance but also for building trust in AI systems, ensuring accountability, and facilitating fair and ethical AI practices.

In this context, we explored three key interpretability techniques: SHAP, LIME, and Captum. SHAP and LIME provide local interpretability by explaining individual predictions, while Captum offers both local and global interpretability. Each of these techniques has its own strengths and unique features. For instance, SHAP values provide a unified measure of feature importance and interaction effects, LIME offers simplicity and flexibility for different kinds of models, and Captum offers a wide range of interpretability algorithms and supports PyTorch models.

In conclusion, employing interpretability techniques such as SHAP, LIME, and Captum can significantly aid in understanding and explaining the predictions made by complex AI models like BERT, thereby making them more transparent and accountable.

References

1. Lundberg, S., Lee, S. (2017). A Unified Approach to Interpreting Model Predictions. ArXiv. /abs/1705.07874

2. Kokhlikyan, N., Miglani, V., Martin, M., Wang, E., Alsallakh, B., Reynolds, J., Melnikov, A., Kliushkina, N., Araya, C., Yan, S. (2020). Captum: A unified and generic model interpretability library for PyTorch. ArXiv. /abs/2009.07896

3. Ribeiro, M.T., Singh, S., Guestrin, C. (2016). "Why Should I Trust You?" Explaining the Predictions of Any Classifier. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1135–1144.

4. Štrumbelj, E., Kononenko, I. (2014). Explaining Prediction Models and Individual Predictions with Feature Contributions. Knowledge and Information Systems, 41, 647–665.

5. SHAP latest Documentation. Available: https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/text.html

6. Rodríguez-Pérez, R., Bajorath, J. (2020). Interpretation of machine learning models using Shapley values: application to compound potency and multi-target activity predictions. Journal of Computer-Aided Molecular Design, 34(10), 1013–1026.

7. Captum: A Unified and Generic Model Interpretability Library for PyTorch [Online]. Available: https://captum.ai/tutorials/Bert_SQUAD_Interpret

8. Captum: A Unified and Generic Model Interpretability Library for PyTorch [Online]. Available: https://captum.ai/tutorials/Bert_SQUAD_Interpret2

9. Captum: Interpretability of IMDB Model [Online]. Available: https://captum.ai/tutorials/IMDB_TorchText_Interpret

10. Captum GitHub Issue 150 [Online]. Available: https://github.com/pytorch/captum/issues/150

11. Kapishnikov, A., Engelhardt, M., Soto, M., et al. (2020). Captum: A Unified and Generic API for Model Interpretability. arXiv preprint arXiv:2009.07813.

12. BERT meets Shapley: Extending SHAP Explanations to Transformer-based Classifiers. Available: https://aclanthology.org/2021.hackashop-1.3.pdf

13. BERT meets Shapley: Extending SHAP Explanations to Transformer-based Classifiers. Available: https://aclanthology.org/2021.hackashop-1.3.pdf

14. Interpreting your deep learning model by SHAP. Available: https://towardsdatascience.com/interpreting-your-deep-learning-model-by-shap-e69be2b47893