Dependable AI (CSL7370)

Implementation of LIME, SHAP & CNN Visualizations

TEAM DESCRIPTION

Rishav Aich (B21AI029)
Pre-final Year (B.Tech. Artificial Intelligence & Data Science)

Tanish Pagaria (B21AI040)
Pre-final Year (B.Tech. Artificial Intelligence & Data Science)

IIT Jodhpur Undergraduates

OVERVIEW

The project aims to enhance understanding of complex models by employing methods such as LIME, SHAP, and CNN visualizations.
Relevant datasets are selected, and the predictions of the models are explained using LIME and SHAP. Furthermore, visualizations are generated using Grad-CAM to facilitate understanding of the models' predictions. The overarching objective is to increase the transparency and comprehensibility of the models.

DATASETS

- **Boston Housing Data (for SHAP & LIME)**
  The dataset concerns the housing prices in the housing city of Boston. The dataset provided has 506 instances with 13 features. Only 9 selected features were used for the problem statement to reduce the high training time.

Column name	Description
INDUS	proportion of non-retail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
RM	average number of rooms per dwelling
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centers
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per $10,000
B	1000(Bk - 0.63)^2 where Bk is the proportion of Black people by town
LSTAT	% lower status of the population
MEDV **(Target Variable)**	Median value of owner-occupied homes in $1000's

- **CIFAR-10 (for Grad-CAM)**
  The dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

SHAP (SHapley Additive exPlanations)

SHAP is a game theoretic approach to explain the output of any machine learning model. The goal of SHAP is to explain the prediction of an instance by computing the contribution of each feature to the prediction.

The SHAP explanation method computes Shapley values from coalitional game theory. The feature values of a data instance act as players in a coalition. Shapley values explain how to fairly distribute the "payout" (the prediction) among the features.

The Shapley value is the average marginal contribution of a feature value across all possible coalitions. The Shapley value is defined via a value function *val* of players in S. The Shapley value of a feature value is its contribution to the payout, weighted and summed over all possible feature value combinations:

$$\phi_j(val) = \sum_{S \subseteq \{1,\ldots,p\} \setminus \{j\}} \frac{|S|!\,(p-|S|-1)!}{p!}\left(val\left(S \cup \{j\}\right) - val(S)\right)$$

where S is a subset of the features used in the model, x is the vector of feature values of the instance to be explained and p the number of features.

Shapley Value Estimation

All possible coalitions (sets) of feature values have to be evaluated with and without the j-th feature to calculate the exact Shapley value. For more than a few features, the exact solution to this problem becomes problematic as the number of possible coalitions exponentially increases as more features are added. Approximation with Monte-Carlo sampling:

$$\hat{\phi}_j = \frac{1}{M}\sum_{m=1}^{M}\left(\hat{f}\left(x^m_{+j}\right) - \hat{f}\left(x^m_{-j}\right)\right)$$

where:

- $\hat{f}(x^m_{+j})$: prediction for x, but with a random number of feature values replaced by feature values from a random data point z, except for the respective value of feature j.
- x^m_{-j}: almost identical to x^m_{+j}, but the value x^m_j is also taken from the sampled z.

Approximate Shapley estimation for single feature value

Required:

Number of iterations M,
instance of interest x,
feature index j,
data matrix X,
machine learning model f

Algorithm:

For all $m = 1, \ldots, M$:

- Draw random instance z from the data matrix X
- Pick a random subset of feature column indices o (with j not in o).
- Construct two new instances:
  - **With j from x: x_{+j},** where all values in x with index in o are replaced by the respective values in z.
  - **Without j from x: x_{-j},** where all values in x with index in o are replaced by the respective values in z and also the value for j is replaced by the value in z.
- Compute marginal contribution:

$$\hat{\phi}_j = \frac{1}{M}\sum_{m=1}^{M}\left(\hat{f}\left(x^m_{+j}\right) - \hat{f}\left(x^m_{-j}\right)\right)$$

Compute Shapley value as the average:

$$\phi_j(x) = \tfrac{1}{M}\textstyle\sum_{m=1}^{M}\phi_j^m$$

Output: Shapley value for the value of the j-th feature

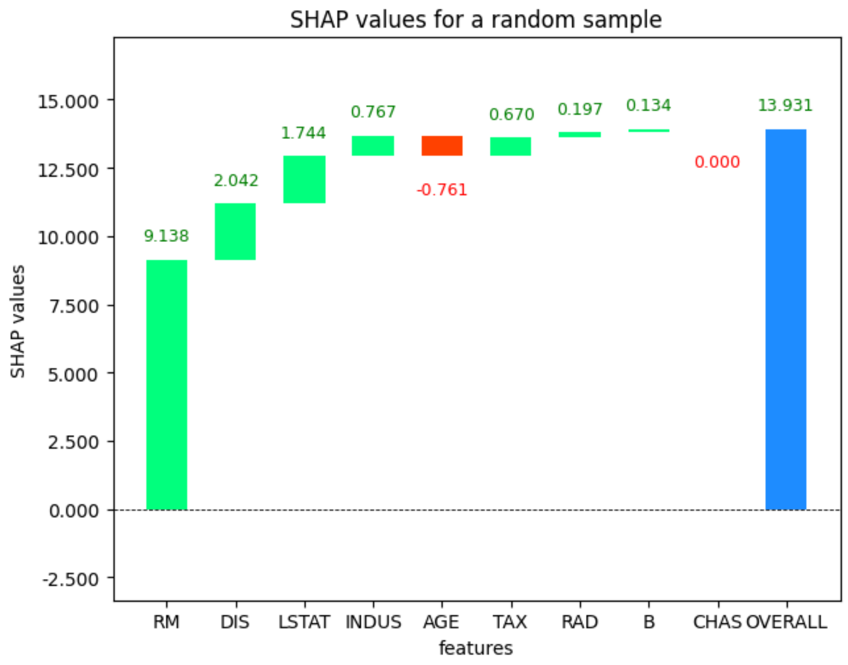
Python Implementation
Methods were implemented for performing the following operations:

- Calculate the worth of the coalition (the mean prediction value of the model for the given coalition of features).
- Generate all possible coalitions of features for a given data point, excluding a specific feature.
- Calculate the contribution of adding a feature to a coalition.
- Generate a list of Shapley values for each data point in the input sample.

Uninterpretable Machine Learning Model (XGBoost Regressor)
XGBoost (eXtreme Gradient Boosting) is a technique where models are built in sequence to correct the errors of the previous ones, improving prediction accuracy.
XGBoost is considered non-interpretable because it combines a large number of decision trees in a complex way, making it hard to understand how individual features influence the model's predictions. This complexity arises from the ensemble of trees, where the final prediction is the sum of the predictions of all trees, and the interactions between these trees are not easily discernible.

XGBoost Regressor was trained on selected features from the Boston Housing Data. The Exact SHAP algorithm was applied by taking 100 data points from the data and calculating the Shapley values for a random sample.

Results
The Shapley values obtained for a random sample from the dataset are shown below:



From the above waterfall chart, the following observations were made:

- The highest positive Shapley value was observed for the average number of rooms per dwelling (RM). It could be easily inferred that the price of housing would increase with more rooms.
- The lowest (and negative) Shapley value was observed for the proportion of owner-occupied units built prior to 1940 (AGE). It could be easily inferred that old housing units would have lower prices.
- The Shapley value for CHAS (Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)) was tending towards zero signifying very little or no contribution to the housing price prediction.

LIME (Local Interpretable Model-Agnostic Explanations)

Local surrogate models are interpretable models that are used to explain individual predictions of black box machine learning models. Surrogate models are trained to approximate the predictions of the underlying black box model. Instead of training a global surrogate model, LIME focuses on training local surrogate models to explain individual predictions.
Mathematically, local surrogate models with interpretability constraint can be expressed as follows:

$$\text{explanation}(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

where:

x: instance.
L: loss (how close the explanation is to the prediction of the original model).
g: explanation model for instance x that minimizes L.
$\Omega(g)$: model complexity, which is kept low.
G: family of possible explanations (models).
π_x: proximity measure (how large the neighborhood around instance x is).

Algorithm for training local surrogate models

- Select the instance of interest for which an explanation of its black box prediction is desired.
- Perturb the dataset and obtain black box predictions for these new points.
- Weight the new samples based on their proximity to the instance of interest.
- Train a weighted, interpretable model on the dataset with the variations.
- Explain the prediction by interpreting the local model.

Python Implementation

Methods were implemented for performing the following operations:

- Generate perturbed samples from a given data point by adding random noise scaled by a specified factor.
- Calculate penalty term for a given model, which is often used in model selection or regularization. For decision tree regressors, it would return the depth of the tree, while for other models (e.g., linear regression), it would return the number of non-zero coefficients.
- Calculate the proximity between two data points using Euclidean distance.
- Compute the loss between a complex model and an interpretable model using perturbed samples while considering model proximity and penalty terms.

A list of simpler interpretable models was created. These models were trained iteratively and the one with the lowest loss compared to the complex model was selected.
The feature importances for the best interpretable model was used for inferring interpretations.

Uninterpretable (Complex) Machine Learning Model (XGBoost Regressor)
XGBoost Regressor was trained on selected features from the normalized Boston Housing Data.

Interpretable (Simple) Machine Learning Models

The following models were selected as explanation models:

- Lasso (alpha=0.3)
- Lasso (alpha=0.7)

  where:

  Lasso: Linear Model trained with L1 prior as regularizer.
  alpha: Constant that multiplies the L1 term, controlling regularization strength.

- Decision Tree Regressor
- Decision Tree Regressor (max_depth=2)
- Decision Tree Regressor (max_depth=3)
- Decision Tree Regressor (max_depth=4)

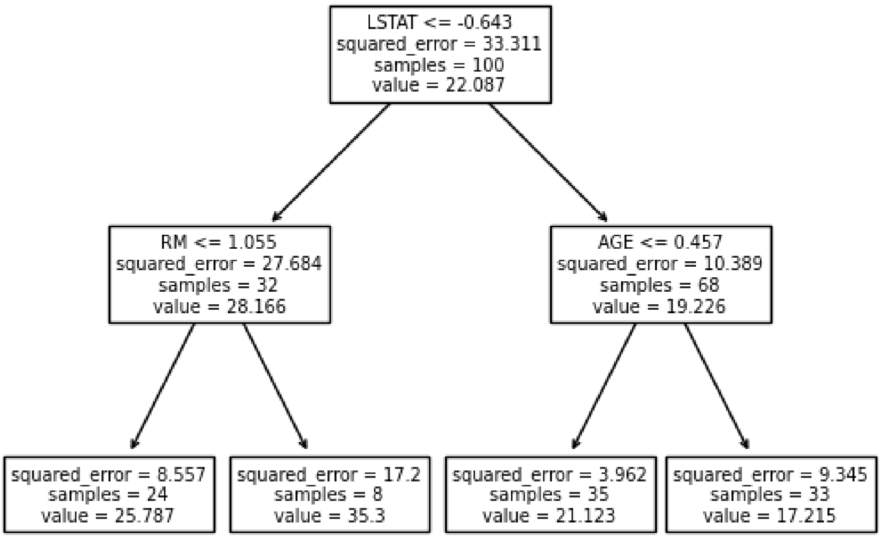
Results
The following loss values were obtained on applying LIME algorithm:

Explanation Model	Loss Value
`Lasso(alpha=0.3)`	`16.105772`
`Lasso(alpha=0.7)`	`10.975143`
`DecisionTreeRegressor(max_depth=2)`	`6.952047`
`DecisionTreeRegressor(max_depth=3)`	`9.172586`
`DecisionTreeRegressor(max_depth=4)`	`10.162239`
`DecisionTreeRegressor()`	`18.816906`

The Decision Tree Regressor model (with max_depth=2 hyperparameter) was selected as the best explanation model. The features importance values for the model are shown below:

AGE	RAD	TAX	DIS	RM	LSTAT	B	INDUS	CHAS
0.102069	0	0	0	0.213658	0.684273	0	0	0

The corresponding decision tree plot for the model is shown below:

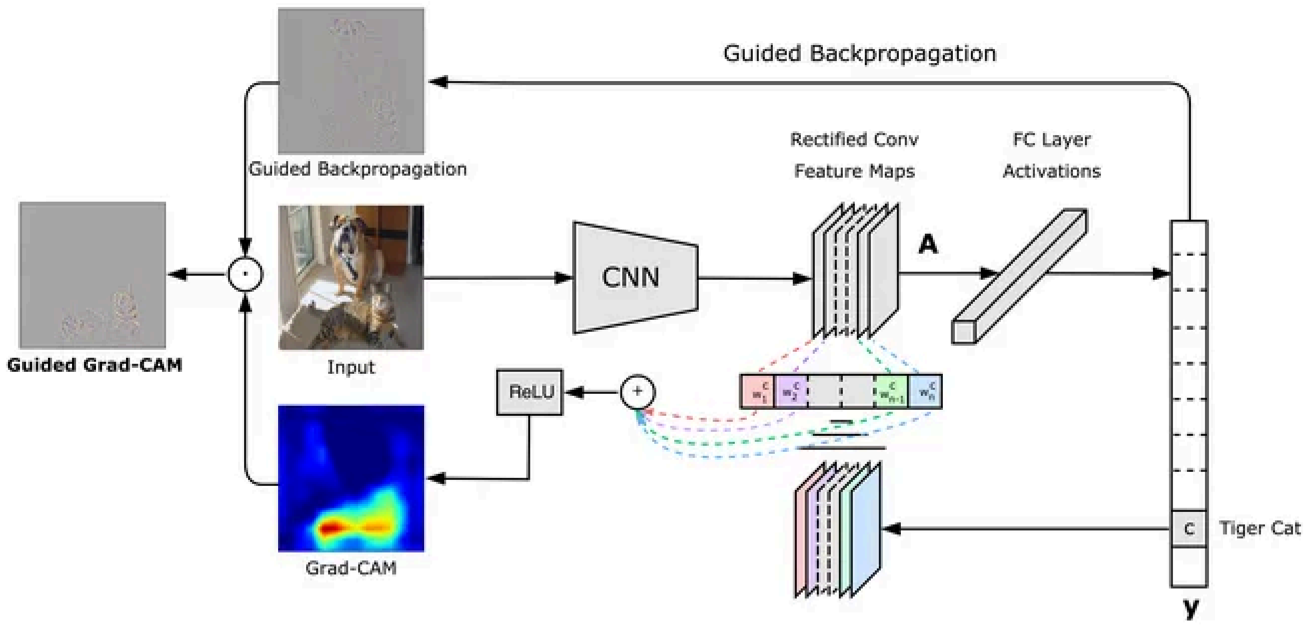


The following observations were inferred from the above:

- The highest feature importance was observed for the feature LSTAT (percentage lower status of the population), followed by RM and AGE.
- All the other features had no contribution to the feature importance of the model.
- The tree shown above is a simpler interpretation for the original model giving a rough estimation on how the features can be adjusted to control the housing prices.

CNN Visualizations using Grad-CAM (Gradient-weighted Class Activation Mapping)

Grad-CAM is a technique used in deep learning to visualize the regions in an image that contribute most to the decision-making process of a convolutional neural network (CNN).
It analyzes gradients in the final convolutional layer to decode feature map importance for a specific class. By generating class-discriminative heatmaps, it identifies and highlights the significant regions in an image contributing to the predicted class, offering insights into the most influential features or patterns. Moreover, Grad-CAM provides spatial localization, pinpointing important areas within the image.



Grad-CAM Architecture

Gradient-weighted Class Activation Mapping (Grad-CAM), uses the gradients of any target concept (say 'dog' in a classification network or a sequence of words in captioning network) flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept.

Approach
In Grad-CAM, preserving the spatial location information of the object is crucial, and this information is lost in a fully connected layer. Thus, the last convolution layer is utilized because its neurons identify parts specific to that class.
To obtain a Grad-Cam of width *u* and height *v* for any class *c*, the gradient of the score for class c, y^c (before the softmax), is computed with respect to feature maps A^k of a convolutional layer, i.e., $\partial y^c / \partial A^k$.

The importance of each feature map k for specific classes is highlighted using the global average pooling technique through the following equation:

$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{-\frac{\partial y^c}{\partial A_{ij}^k}}_{\text{Negative gradients}}$$

where summation over i and j refer to global average pooling and partial differentials refer to the gradients over backpropagation.

After obtaining the gradients, a combination of activation maps is performed, followed by ReLU:

$$L_{GRAD-CAM}^c = RELU \sum_k a_k^c A^k$$

ReLU highlights positive influences on the class. Without it, localization maps may include unwanted information, like negative pixels from other categories, affecting performance.

The class score for a particular class *c* is computed as:

$$S^c = RELU \sum_i \sum_j \sum_k w_k^c A_{ij}^k$$ obtained from $$S^c = \sum_k w_k^c 1/Z \sum_i \sum_j A_{ij}^k$$

Algorithm for implementing Grad-CAM

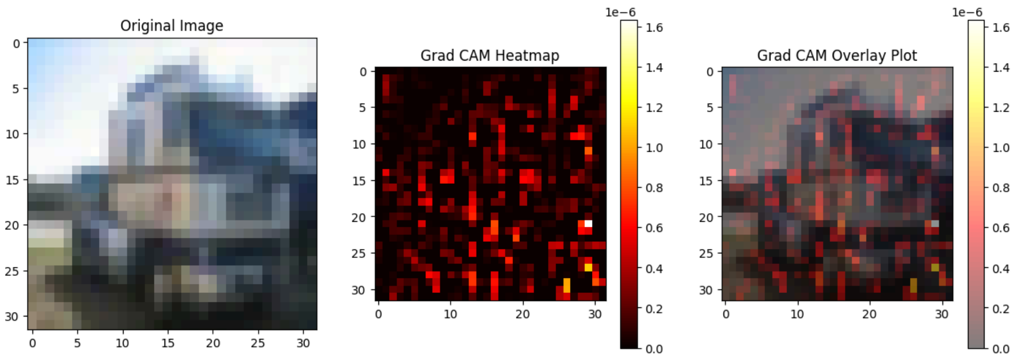
- Train a CNN model or load a pre-trained model capable of making predictions on the target image.
- Preprocess the input image.
- Forward pass: Pass the preprocessed image through the model to obtain the class scores (output of the final classification layer).
- Use backpropagation to compute the gradients of the target class scores with respect to the output feature map (the last convolutional layer before the classification layer).
- Compute GradCAM: Multiply the gradients obtained in the backward pass with the feature map of the last convolutional layer. Sum these products across all channels to obtain a coarse map of class activation.
- Resize the coarse map to the original image size to overlay it on the image to highlight the regions that contributed most to the model's decision.

Python Implementation

The classification model was a Convolutional Neural Network (CNN) composed of several layers tailored for image classification tasks. It commenced with three convolutional layers followed by max-pooling layers to extract salient features from the input image. Subsequently, the output was flattened and propagated through two fully connected layers for classification. To mitigate overfitting, a dropout layer was incorporated during training. Finally, the softmax activation function was employed to produce class probabilities as the output. The model was compiled with categorical cross-entropy loss, categorical accuracy metric, and Adam optimizer.
The model was trained on the 32x32 images from the CIFAR-10 dataset.

The heatmap highlighting important regions in an image for the specified class prediction was calculated. The gradients of the predicted class score with respect to the feature maps of a specified layer were computed. ReLU activation function was applied on the gradients to produce the activation map.
The heatmap was overlaid onto the original image. A colormap was applied to the heatmap for better visualization and blends it with the original image using adjustable transparency.

Results for a sample image from the dataset



CNN Visualization of a random image sample from the dataset with Grad-CAM heatmap overlay

In the above visualization, it could be easily observed that the model was able to identify the distinct features of the truck ignoring its surrounding background. The body outline was distinctly highlighted in this case. The contribution of the various pixels (components) of the image was demonstrated effectively allowing the users to understand which parts of the image were focused on by the model for making predictions, aiding in interpretability.

REFERENCES

SHAP and Shapley Values
https://arxiv.org/pdf/1705.07874.pdf
https://christophm.github.io/interpretable-ml-book/shap.html
https://towardsdatascience.com/shap-explained-the-way-i-wish-someone-explained-it-to-me-ab81cc69ef30
https://www.depends-on-the-definition.com/shapley-values-from-scratch/

LIME and Local Surrogate Models
https://arxiv.org/pdf/1602.04938.pdf
https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/
https://christophm.github.io/interpretable-ml-book/lime.html
https://www.depends-on-the-definition.com/debugging-black-box-text-classifiers-with-lime/

Grad-CAM for CNN
https://arxiv.org/pdf/1610.02391.pdf
https://medium.com/@ninads79shukla/gradcam-73a752d368be