Introduction

In this assignment you will explore the theory and applications of SHAP. As some portions of the assignment require running models on GPUs, all starter code for the programming portions are provided in a Google Colab notebook here. We highly recommend saving a copy of this notebook on your own drive and running the code on Google Colab which gives you free access to GPUs. Note that you must click "File > Save a copy .." in order to save your edits to the notebook and run the code. When running your code, make sure you are opting to use GPUs by selecting the GPU option in "Runtime > Change runtime type".

Submit a write-up using this LaTeX template and submit a compressed .zip file of your .ipynb notebook. Please start write-ups for each problem on a new page.

SHapley Additive exPlanations (SHAP)

Let $X, Z$ be the original input space and simplified input space, respectively. Further let $h_x : Z \rightarrow X$ be a mapping function specific to an input $x$ which recovers the original inputs from the simplified inputs. For NLP tasks $x \in X = \mathbb{R}^n$ may be a bag of words feature vector (where $n$ is the total number of possible words) and $z \in Z = \{0,1\}^n$ may be a vector of 1’s and 0’s where a dimension for a word has value 1 if the original input $x$ has a non-zero count for that word and 0 otherwise. $h_x(z)$ then maps feature dimensions with value 1 in $z$ to the original word count other dimensions to 0. In words, $z$ simply captures the presence/absence of words in a sentence while throwing out information about the frequency of their occurrence. $z$ may be viewed as meaningful “compression” of $x$ even though $h_x(z) = x$ since $h_x$ is a function specific to a single $x$.

Let $f : \mathcal{X} \rightarrow \mathbb{R}$ be the original prediction model (which usually belongs to a complex model class), such as a deep transformer neural network that performs sentiment classification of text sentences, and $g_x : Z \rightarrow \mathbb{R}$ be a local explanation model which is an interpretable approximation of the original model around the neighborhood of $x$. Local explanation models seek to satisfy

$$z' \approx z \Rightarrow g_x(z') \approx f(h_x(z')).$$

where $x = h_x(z)$. Roughly speaking, $g_x(z)$ attempts to fit the first-order Taylor approximation of $f(h_x(z))$ around $z$. For $\mathcal{X} = \mathbb{R}^n$, a common choice is to have $Z = \{0,1\}^n$ and an explanation model $g_x$ that is linear:

$$g_x(z) = \phi_0(f, x) + \sum_{i=1}^n \phi_i(f, x)z_i.$$  

Intuitively, $z$ is an indicator vector which only captures the presence/absence of features and $\phi_i(f, x)$ is a measure of how much the inclusion of feature $i$ leads to an increase in $f(x)$, and hence is representative of the importance of feature $i$ in the prediction $f(x)$.

One might wonder if such a linear explanation model $g_x$ is uniquely identifiable given $f$ and $h_x$. An astonishing result from Cooperative Game Theory is that a linear local explanation model $g_x$ which satisfies Local accuracy,
Problem 2. Implementing Linear SHAP [40 points]

Let our original prediction model itself be a linear model

\[ f_\theta(x) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i \]

Further assume that we have feature independence so that all dimensions of \( x \) are independent. Show that equation 1 and equation 2 can be rewritten as

\[ \phi_0(f, \bar{x}) = f(\mathbb{E}_{x \sim p(x)}[x]) \]
\[ \phi_i(f, \bar{x}) = \theta_i (\bar{x}_i - \mathbb{E}_{x \sim p(x)}[x_i]) \text{ for } i = 1, \ldots, n \]

Problem 2. Implementing Linear SHAP [40 points] The IMDB dataset has been loaded for you in the provided starter code. The inputs \( x \) are normalized bag of words representations of movie review sentences and targets \( y \) represent binary category labels “positive” and “negative”. Recall that a logistic regression model for binary classification is a parameterized function that outputs the Bernoulli probability

\[ \frac{1}{1 + e^{f_\theta(x)}} \]

where \( f_\theta(x) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i \) is a linear model.

(a) [5 points] Fit a logistic model on the training set using sklearn.linear_model.LogisticRegression with arguments penalty=’l2’, C=0.1. Report the accuracy on the test set using the .score() method.

(b) [20 points] Compute the Linear SHAP value for the first ten features of the first test input \( \text{X\_test}[0, :10] \) using equation 3 and equation 4. To estimate the expectations \( \mathbb{E}_{x \sim p(x)}[x], \mathbb{E}_{x \sim p(x)}[x_i] \), use the empirical mean
of the feature vectors in the training set. (*Hint: you should be computing a total of 11 values = 1 base value + 10 for each feature*)

(c). [15 points] Now we use a standardized library to compute SHAP values. Use `shap.LinearExplainer` to compute the SHAP values. For the `model` argument, pass in our fitted logistic model. For the `masker` argument, pass in a tuple (mean, cov). Here, mean is the mean of the feature vectors in the training set we used before. For cov you may simply pass in a dummy value 0 as we are assuming feature independence. Report the SHAP values for the first ten features of the first test input `X_test[0, :10]` as we did in the previous problem using the `shap_values()` method. How do they compare to your manually computed SHAP values from part (b)?

**Problem 3. SHAP with Transformers [20 points]** This time our original predictor will be a pretrained deep transformer network trained on a similar Sentiment Analysis task. Inputs to the model are sentences and our transformer attempts to predict the top $K$ emotions best labeling the sentence. As our original model is more complicated we will have to compute the SHAP values using a different approximation than LinearSHAP. Use `explainer = shap.PartitionExplainer` with the argument `model=pred, masker=tokenizer`. Compute the SHAP values on sample text by directly passing `sample_data` (defined in the starter code) to the explainer. Visualize the SHAP values using `shap.plots.text`. Attach the visualizations in your write-up and comment on your findings.

**Problem 4. SHAP with ConvNets [20 points]** This time our original model will be a ConvNet trained for MNIST image classification. Use `shap.DeepExplainer` with arguments `model=model, data=background`. Compute the SHAP values on `test_images` (defined in starter code) using `.shap_values()` method of the explainer. Visualize the explanations using `shap.image_plot`. Attach the visualizations in your write-up and comment on your findings. (*Hint: you may need to use np.swapaxes to transpose your image dimensions to get proper visualizations*)