Homework #1

CSEP 590B: Explainable AI
Prof. Su-In Lee
Due: 4/25/22 11:59 PM

1 Removal-based explanations (15 points)

(a) [1 points] In the framework for removal-based explanations, what are the three implementation choices required for each method?

(b) [3 points] What are the specific design choices made by RISE (Petsiuk et al., 2018)?

(c) [3 points] What are the specific design choices made by SAGE (Covert et al., 2020)?

(d) [1 points] Which of the three design choices determines whether an explanation describes *local* or *global* feature importance?

(e) [3 points] Describe two benefits each for using local versus global explanations.

(f) [4 points] Why is SHAP sometimes viewed as a special case of LIME? **Hint:** focus on the summarization technique, the third implementation choice.

2 Shapley value properties (20 points)

For a set of d players represented by $D = \{1, \ldots, d\}$ and a cooperative game $v : 2^d \mapsto \mathbb{R}$, the Shapley value for each player $i \in D$ is defined as:

$$\phi_i(v) = \sum_{S \subseteq D \setminus \{i\}} \frac{|S|!(d - |S| - 1)!}{d!}(v(S \cup \{i\}) - v(S)). \tag{1}$$

(a) [3 points] Prove the *null player* axiom of the Shapley value, which states that if a player contributes no value in a game $v : 2^d \mapsto \mathbb{R}$, or $v(S \cup \{i\}) - v(S) = 0$ for all $S \subseteq D \setminus \{i\}$, then $\phi_i(v) = 0$.

(b) [3 points] Prove the *linearity* axiom of the Shapley value, which states that given two cooperative games $u : 2^d \mapsto \mathbb{R}$ and $v : 2^d \mapsto \mathbb{R}$ and a third game w defined as $w(S) = u(S) + v(S)$ for all $S \subseteq D$, the following holds for all players $i \in D$:

$$\phi_i(w) = \phi_i(u) + \phi_i(v).$$

(c) [4 points] Prove the *efficiency* axiom of the Shapley value, which states that the following holds for all games $v : 2^d \mapsto \mathbb{R}$:

$$\sum_{i=1}^{d} \phi_i(v) = v(D) - v(\emptyset).$$

Hint: recall the Shapley value's ordering interpretation (see Problem 3), and first show that the efficiency property holds when we consider only a single ordering.

S	$\emptyset$	$\{1\}$	$\{2\}$	$\{3\}$	$\{1,2\}$	$\{1,3\}$	$\{2,3\}$	$\{1,2,3\}$
$v(S)$	0	1	1	1	2	2	2	3

(d) [3 points] Calculate the Shapley value for all players $i \in \{1,2,3\}$ in the following cooperative game $v(S)$:

(e) [3 points] Calculate the Shapley value for all players $i \in \{1,2,3\}$ in the game $v(S)$ given by

$$v(S) = x_1 + 2x_2 + 3x_3,$$

where x_i are binary variables that are equal to 1 if $i \in S$ and 0 otherwise.

(f) [4 points] Calculate the Shapley values for all players $i \in \{1,2,3,4,5\}$ in the game $v(S)$ given by

$$v(S) = 1x_2 + 1x_3 + 2x_4 + 3x_1x_3 + 5x_2x_5 - 10x_1x_2x_4,$$

where x_i are binary variables that are equal 1 if $i \in S$ and 0 otherwise. **Hint:** write a Python function to calculate the Shapley value automatically, using either the formula in Eq. 1 or Eq. 2.

3 Shapley value estimation (20 points)

The Shapley values $\phi_1(v), \ldots, \phi_d(v)$ for a game $v : 2^d \mapsto \mathbb{R}$ can also be expressed in terms of player orderings. Let π denote an ordering of the indices $\{1, \ldots, d\}$, let $\pi^{-1}(i)$ represent the position of i in the ordering, and Π be the set of all such orderings. For example, with $d = 2$ the only possible orderings are $\pi = [1,2]$ and $[2,1]$, so we have $\Pi = \{[1,2],[2,1]\}$, and given $\pi = [2,1]$ we have $\pi^{-1}(1) = 2$ and $\pi^{-1}(2) = 1$.

The Shapley value for the ith player is then given by:

$$\phi_i(v) = \frac{1}{d!} \sum_{\pi \in \Pi} v\big(\{j : \pi^{-1}(j) \leq \pi^{-1}(i)\}\big) - v\big(\{j : \pi^{-1}(j) < \pi^{-1}(i)\}\big). \tag{2}$$

We will first prove that this equation is equivalent to the one in Eq. 1, and we will then consider how to use this formulation to approximate Shapley values when d is large.

(a) [2 points] Given an ordering π, what do the sets $\{j : \pi^{-1}(j) < \pi^{-1}(i)\}$ and $\{j : \pi^{-1}(j) \leq \pi^{-1}(i)\}$ represent?

(b) [2 points] Define $\Delta(v, \pi, i) = v\big(\{j : \pi^{-1}(j) \leq \pi^{-1}(i)\}\big) - v\big(\{j : \pi^{-1}(j) < \pi^{-1}(i)\}\big)$. What does $\Delta(v, \pi, i)$ represent?

(c) [4 points] Fix a player $i \in D$ and a subset $S \subseteq D \setminus \{i\}$. How many orderings are there in the set Π, and in how many of these is i preceded by exactly the set of indices in S? **Hint:** the indices in S can appear in any order as long as they all precede i.

(d) [4 points] Using the above, prove that the formula in Eq. 2 is equivalent to Eq. 1. **Hint:** try grouping the terms in Eq. 2 by the subset preceding i.

(e) [4 points] Shapley values are impractical to calculate for games with many players (large values of d), so we often resort to approximations. One simple approach involves sampling n orderings $\pi_1, \ldots, \pi_n \in \Pi$ independently at random, and then calculating estimates for each Shapley value as follows:

$$\hat{\phi}_i(v) = \frac{1}{n} \sum_{j=1}^{n} \Delta(v, \pi_j, i).$$

Prove that $\hat{\phi}_i(v)$ is an unbiased estimator of $\phi_i(v)$, or that $\mathbb{E}[\hat{\phi}_i(v)] = \phi_i(v)$. **Hint:** the randomness in $\hat{\phi}_i(v)$ comes from the sampled orderings $\pi_1, \ldots, \pi_n$, and each possible ordering $\pi \in \Pi$ has probability $p(\pi) = \frac{1}{d!}$ because they are chosen at random.

(f) [4 points] For a single ordering $\pi \in \Pi$ sampled at random, denote $V_i = \text{Var}(\Delta(v, \pi, i))$. What is $\text{Var}(\hat{\phi}_i(v))$ in terms of V_i? What does this suggest about our estimator as the number of sampled orderings n becomes large? **Hint:** see the similar question from HW0.

4 Permutation tests (20 points)

A permutation test is a removal-based explanation that measures the impact on the dataset loss from removing individual features. The name comes from from the fact that features are removed by permuting (randomly shuffling) columns of the dataset, where each column corresponds to a single feature. Here, we'll implement permutation tests from scratch.

For this problem we'll use the census income classification dataset, where the task is to predict if an individual has an income above $50,000. A nicely formatted version of the dataset can be loaded from the SHAP package, which you can install into your Python package manager (e.g., Anaconda, virtualenv) as follows:

```
pip install shap
```

Once the SHAP package is installed, load the dataset and split it into a train and test set with the following code:

```
import shap
from sklearn.model_selection import train_test_split

X,y = shap.datasets.adult()  # Numerical version of data
X_display, y_display = shap.datasets.adult(display=True)  # Human-readable data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)
```

(a) [4 points] Train a gradient boosting classifier using `sklearn` as follows (please use this exact code):

```
from sklearn.ensemble import GradientBoostingClassifier

clf = GradientBoostingClassifier(n_estimators=100, random_state=10)
clf.fit(X_train.values, y_train)
```

Report the zero-one classification error (using a classification threshold of 0.5) and log-loss for both the train and test sets.

(b) [4 points] Implement the permutation test approach for the trained model. For each feature, measure the change in the test set zero-one error after permuting the corresponding feature column. Visualize the importance values with a bar plot.

(c) [4 points] Permutation tests are inherently stochastic because they permute features randomly. Run the permutation test from part (b) 10 times and visualize the variance in the importance values using a bar plot with standard deviation error bars.

(d) [4 points] One possible variation of the permutation test is to change the feature removal approach. Implement a new method similar to (b) that removes features by setting them to their mean instead of permuting them. Visualize the importance values using a bar plot.

(e) [4 points] Another possible variation is to change the model behavior we measure. Implement a new method similar to (b) that measures the change in the test set log-loss rather than the zero-one error. Visualize the importance values using a bar plot.

5 Popular feature importance packages (25 points)

In this problem, we'll use LIME and SHAP to explain a tree-based model. LIME and SHAP are widely used packages to calculate feature importance for ML models. SHAP implements multiple explanation methods (all variants on Shapley values), and we will use two of them in this problem. You'll need to install both packages as follows, preferably using a recent version of Python 3:

```
pip install shap
pip install lime
```

Once again, this problem will use the census income classification dataset. Load the data in the same way as in Problem 4, using the exact same train/test split. In addition, please use the same model you trained in Problem 4(a).

(a) [10 points] Compute local attributions for the first 200 samples from the test dataset (`X_test[:200]`). We will call these samples *explicands* since they are the samples being explained. Use the following methods (and note that KernelSHAP and LIME may take a while to run):

- **TreeSHAP**. Example:

```
import shap

explainer = shap.TreeExplainer(clf, baselines)
attributions = explainer.shap_values(explicands)
```

- **KernelSHAP**. Example:

```
import shap

explainer = shap.KernelExplainer(clf.predict_proba, baselines)
attributions = explainer.shap_values(explicands)[1]
```

- **LIME**[1] Example:

```
import lime.lime_tabular

explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values)
attribution = explainer.explain_instance(explicand, clf.predict_proba).local_exp[1]
# Note: you must reshape the attribution and run explain_instance for every explicand
```

Note that both TreeSHAP and KernelSHAP require a set of *baselines*, which are used to replace held-out features with samples from their marginal distribution. For this problem, use the last 100 samples from the test set (`X_test[-100:]`) as the baselines.

Visualize the results for each method using three separate summary plots: (`shap.summary_plot`):

```
shap.summary_plot(attributions, explicands)
```

Based on these summary plots, what are the most important features according to each method?

(b) [5 points] Visualize the attributions from each method for the features "Age" and "Hours per week" with a scatter plot, putting the feature value on the x-axis and the feature attribution on the y-axis. Use the `matplotlib` library to generate the plots. Based on these plots, what observations can you make about how the these features relate to the model's income prediction?

(c) [5 points] Visualize the attributions from each method for the sample with index 1 in the test set (`X_test.iloc[1]`) using a bar plot, with the feature names on the y-axis and the feature attributions on the x-axis . Based on the TreeSHAP attributions, describe one feature that makes a strong positive contribution to this sample's prediction and one feature that makes a strong negative contribution.

(d) [5 points] An important parameter for KernelSHAP is the number of samples (the `nsamples` argument in `KernelExplainer`'s `shap_values()` function). Using more samples will improve the quality of the final

[1] Note that LIME only supports explaining a single explicand at a time, so it must be run separately for each sample.

feature attribution estimates, but it is more costly.[2] For the previous parts of this problem, we used the default setting of this parameter (see here).

To investigate this parameter, run KernelSHAP for the sample with index 1 in the test set (`X_test.iloc[1]`) with `nsamples=10`, `nsamples=100` and `nsamples=1000` for 10 runs each. Once again, use a bar plot to depict the mean local feature attributions, but this time include error bars to depict standard deviation across runs of the local feature attributions for each number of samples. Explain the result.

[2]TreeExplainer does not have this parameter because it computes the marginal Shapley values exactly.