Progress report - Week 2 - 9/09 - 9/16

Parul Gupta (pargupta@umass.edu)

September 14, 2020

1 Milestones Planned

The following milestone was planned for this week:

1. Implement equation parsing into parse tree.

2 Milestones Achieved

2.1 Implement equation parsing into parse tree

2.1.1 STATUS: Done

2.1.2 Things done here

1. Worked through the theory behind the equation/constraint for fairness in machine learning.
2. Implemented Python code to parse the equation and added tests for the same.

2.1.3 Brief methodology

We are trying to implement fairness in machine learning for classification (might extend it to regression later).
To apply Seldonian algorithm[1] on fairness, we need to define some terminology and definitions to be able to come up with a mathematical expression used further to implement the code.
Definition/terminology:

1. θ: Model parameters.
2. $g(\theta)$: Fairness constraint defined by the user. The tutorial [2] on fairness describes popularly used fairness constraints which can be considered for $g(\theta)$ to measure the fairness of the estimator.
3. Base variables: the constants and the classes such as TP - true positive, TN - true negative, FP - false positive, FN - false negative.

4. $z(\theta)$: The true distribution of each node of the expression tree (i.e, base variables as well as true distribution of the sub-expressions of the constraint).
5. $\hat{z}(\theta, D)$: The unbiased estimates of $z(\theta)$ on the data set, D.

We define the classes as:

$$FP(A) = \begin{cases} 1 & if\ \theta\ gives\ false\ positive\ on\ X_i,\ Y_i, \\ 0 & otherwise. \end{cases}$$

The above will be defined for only the $T_i = A$ as it is mentioned for FP(A). Consider the value as '$None'$ for other groups.

Derivation of expectation of of a class:
Suppose we are calculating for $FP(A)$.
Assume X to be an indicator function defined only in case type=A as

$$x_i = \begin{cases} 1 & if\ FP\ occurred\ for\ ith\ datapoint, \\ 0 & otherwise. \end{cases}$$

Our data samples can be assumed to be independent and identically distributed.
Our estimate of prbability of a datapoint being an FP, $\hat{p} = 1/n * \sum(x_i)$.
We can safely assume this to be binomial random variable.

$$E[\hat{p}] = 1/n * np = p$$

As we do not know p, we approximate it to $\hat{p}$.

To begin with the implementation, we need to create an expression tree which can parse the constraint, $g(\theta)$, provided by the user. After parsing it into an expression tree, we need to evaluate the estimate of the expression under the data set (X, Y, T). We, finally, find the 95% confidence interval around the estimate (say, using t-test) by evaluating a confidence interval at every node of the expression tree through bound propagation. We can safely say that the constraint is satisfied if this confidence interval lies below 0.

2.1.4 Code and results

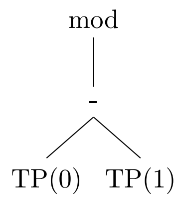
Input: Reverse polish notation of an equation with the following valid elements:

1. **Operators**: addition (+), subtraction (-), multiplication (*), division (/), power ($\hat{}$) and mod (|).
2. **Operands**: constants (real numbers), classes (TP - true positive, TN - true negative, FP - false positive, FN - false negative)
3. **Delimiter**: space(' ')

True Positive Rate Sample:

```
TP(0) TP(1) - mod
```

where $0 = female$ and $1 = male$.
The code (*construct_expr_tree*) parses the reverse polish notation equation and form an expression tree.
Example of the above sample will be:



Now, for evaluation, we will pass data in the form of true value, Y, predicted value, $\hat{Y}$ and sensitive attribute, T. All these will be *numpy.Series* (1-D array). The code (*eval_expr_tree*) will find the $z(\theta)$ of the whole expression for the data provided and return the result estimate of $g(\theta)$. Y and $predicted_Y$ are assumed to be 0,1 binary classification.

Test runs:

```
expression = 2 3 - | 5 *
Output of the above evaluation = 5.0

Y = [0, 0, 0, 1, 1, 1]
pred_Y = [1, 1, 1, 1, 1, 1]
T = [0, 1, 0, 1, 0, 1]
expression = TP(0) TP(1) - |
Output of the above evaluation in terms of estimate = 0.3333333333333333
```

The output is as expected -> $\mathbb{E}[TP(0)] = 0.33$, $\mathbb{E}[TP(1)] = 0.66$, thus, $\mathbb{E}[|TP(0) - TP(1)|] = 0.33$.
I have setup a private github repo where I intend to put all the code: fair-work

Next steps for next week: Implement confidence interval bound propagation for the parsed tree, instead of just the estimate.

References

[1] Philip S Thomas, Bruno Castro da Silva, Andrew G Barto, Stephen Giguere, Yuriy Brun, and Emma Brunskill. Preventing undesirable behavior of intelligent machines. *Science*, 366(6468):999–1004, 2019.

[2] Ziyuan Zhong. *A Tutorial on Fairness in Machine Learning*, 2018.