User-centric Systems for Data Science **Prof. John Liagouris**
Fall Semester 2020 liagos@bu.edu

Assignment 3

Submission deadline: November 13 2020, 11:59 pm

This assignment is based on a data science challenge from HackerEarth and requires building a Machine Learning pipeline. For this assignment, you will use (i) your operator library from Assignments 1 & 2 to perform some ETL operations, (ii) a Machine Learning library (scikit-learn) to train and test a (black-box) model, and (iii) the InterpretML toolkit to explain the model predictions. Below is a description of the challenge.

A leading affiliate network company wants to leverage Machine Learning to improve their conversion rates and eventually their topline. Their network is spread across multiple countries in Europe such as Portugal, Germany, France, Austria, Switzerland, etc. Affiliate network is a form of online marketing channel where an intermediary promotes products/services and earns commission based on conversions (click or sign up). The benefit companies see in using such affiliate channels is that they are able to reach audiences that do not exist in their marketing reach.

The company wants to improve their CPC (cost per click) performance. A future prediction about an ad performance will give them enough headstart to make changes (if necessary) in their upcoming CPC campaigns. The company plans to use a state-of-the-art model for this purpose to achieve high prediction accuracy. However, this model is a black-box and the company cannot take its predictions into account in decision making without convincing explanations. Your solution must provide meaningful explanations to individual predictions as well as insights into the model itself.

The data you will use for the assignment can be found here. This dataset contains more than 12M records that are anonymized to protect the privacy of individuals. For the purposes of the assignment, it is sufficient to use 100K randomly selected records: 70K for training and 30K for testing.

1. Data schema 3

2. TASK I: Basic statistics (credits: 20/100) 3

3. Task II: ETL operations (credits: 30/100) 4

4. TASK III: Model training and validation (credits: 10/100) 4

5. TASK IV: Inference and explanations (credits: 40/100) 4

6. Testing 6

7. Logging 7

8. Git 7

9. Deliverables 7

10. Resources 7

1. Data schema

The data we will use for this assignment are given in a CSV file with the schema shown in Table 1. The schema includes nine attributes, five categorical and four numerical. These are not *cleaned* data (cf. Tasks I & II), that is, some values are missing from the CSV file, while others may appear in "variations" across different records (e.g. values "`IE`" and "`Internet Explorer`" of attribute `browserid` both refer to the same browser type). The last attribute (`click`) is the variable we want to predict (Task III) and later explain (Task IV).

Attribute	**Description**
`ID`	A unique ID for each record (`string`)
`datetime`	Record timestamp (`MM-DD-YYYY HH:MM:SS`)
`siteid`	Website ID (`int`)
`offerid`	Offer ID (`int`)
`category`	Offer category (`int`)
`merchant`	Seller ID (`int`)
`countrycode`	Country where affiliates reach is present (`char`)
`browserid`	Browser used (`string`)
`devid`	Device used (`string`)
`click`	Denotes whether the user clicked on the Ad or not (`boolean`)

Table 1: Data Schema

2. TASK I: Basic statistics (credits: 20/100)

The first task is to compute some basic statistics about your dataset using your operator library. In particular, you must compute the number of:

A. Missing values for each attribute of Table 1

B. Distinct values for attributes `countrycode`, `browserid`, and `devid`

and print the result to standard output as a list of tuples of the form `[(att_name, number))]`

Hint: Consider adding a `DISTINCT` operator to your operator library.

3. Task II: ETL operations (credits: 30/100)

The second task is to use your operator library to perform some ETL (Extract-Transform-Load) operations that are necessary in order to train your model. Specifically, you must transform:

A. All missing values to zero.

B. The given datetime values of the form `MM-DD-YYYY HH:MM:SS` to four new attributes `month`, `day`, `hour`, and `minute` (omit `year` and `seconds`).

C. The three attributes `countrycode`, `browserid`, and `devid` from categorical to numerical by mapping each distinct value to a unique number (`int`). You should try "cleaning" your data so that, for example, values "`Firefox`" and "`Mozilla Firefox`" are mapped to the same number.

In addition to the previous transformations, you must also project out the attribute `ID` before passing the data to the classifier in Task III.

Hint: Consider adding a `MAP` operator to your library that applies a user-defined function (data transformation) on a tuple-by-tuple basis. Avoid building an ad-hoc operator for each transformation.

4. TASK III: Model training and validation (credits: 10/100)

The third task is to train a tree-based model using the Light Gradient Boosting Machine framework on the preprocessed data from Task II. To this end, you first need to transform the data into an array-like data structure that the LGBM classifier expects. After training your classifier, you must validate and report its accuracy using the test data.

Hint #1: You may want to have a look at `train_test_split()` method of scikit-learn that provides an easy way to randomly split your dataset in two parts: one for training and one for testing.

Hint #2: You may want to use the `classification_report()` method of scikit-learn to test and report the accuracy of your trained model. Alternatively, you may use the ROC module of InterpretML to generate the Receiver Operating Characteristic Curve.

5. TASK IV: Inference and explanations (credits: 40/100)

The fourth task is to use two interpretability techniques (LIME and SHAP) from the InterpretML toolkit to help users understand what your model has learned and how it makes its predictions. This will build trust in your model and encourage the affiliate company to use it in critical decision making.

This task requires generating both *local* and *global* explanations. The former help users interpret individual predictions whereas the latter provide insights into the model itself. For local explanations, you must use randomly selected instances from the given CSV file. Make sure that the instances you use for predictions are not included in the 100K instances you used to train and validate your model.

Local explanations

The first sub-task is to explain two individual predictions using LIME, the technique we discussed in Lecture 11. The predictions for the two instances you pick must correspond to different classes. For each prediction, the expected explanation is a plot like the one shown in Fig. 1.

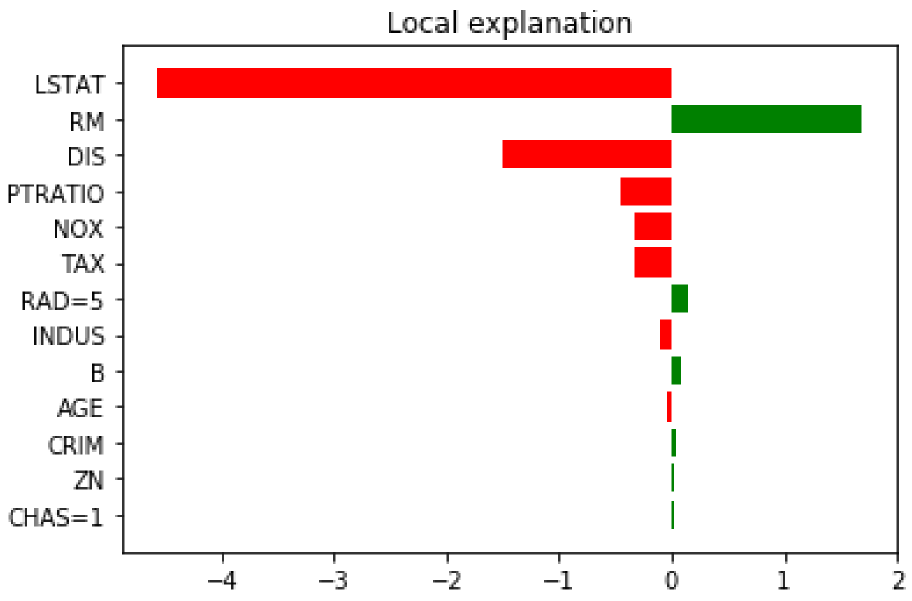

Fig. 1: Example of a local explanation using LIME on a model that predicts house prices in Boston. The y-axis shows the features of the linear model and the x-axis shows the effect (i.e. feature value times feature weight) of each feature to the prediction for the particular instance used. Red and green bars denote negative and positive effects respectively.

The second sub-task is to explain the same individual predictions using SHAP, the technique we discussed in Lecture 14. Specifically, you must use SHAP to generate two so-called *force plots* (one for each prediction) like the one shown in Fig. 2.

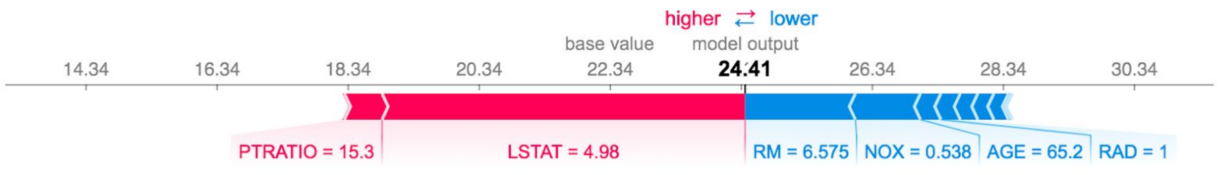

Fig. 2: Example of a local explanation using SHAP on a model that predicts house prices in Boston. Each "force" (arrow) corresponds to a feature value that "pushes" to decrease (blue) or increase (red) the average prediction. The length of an arrow is the absolute Shapley Value.

Global explanations

The third sub-task is to generate a global explanation for your model using the sub-modular pick method of LIME. The explanation in this case must be a list of 10 "representative" instances (cf. Lecture 11) along with their predictions.

The fourth and final sub-task is to generate an alternative global explanation for your model using a so-called SHAP *summary plot* like the one shown in Fig. 3.

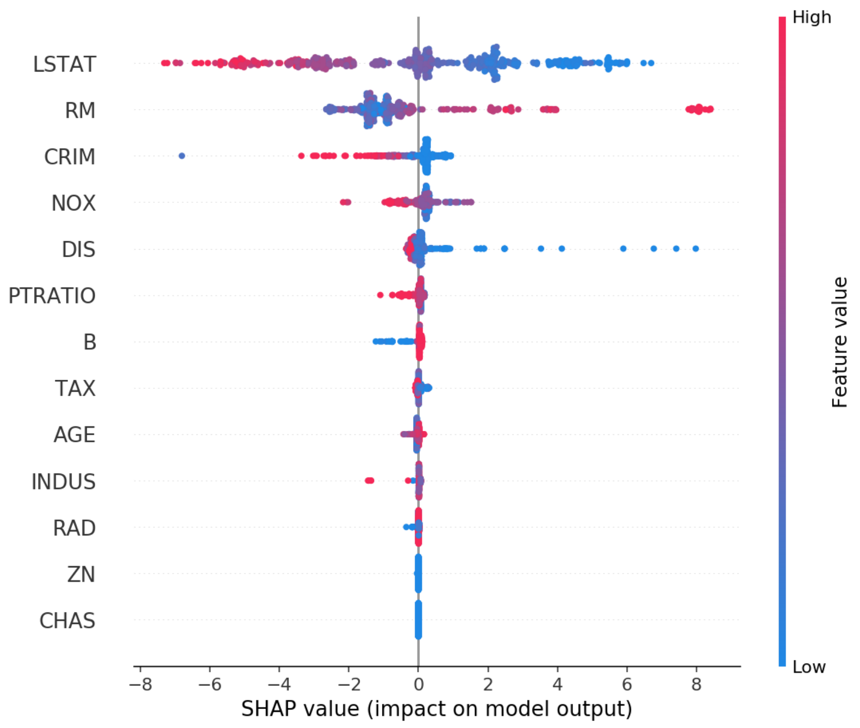

Fig. 3: Example of SHAP's summary plot on a model that predicts house prices in Boston. Each point in the plot corresponds to the Shapley Value of a feature value in a particular instance. The y-axis shows the features (ordered by their importance) and the x-axis shows the Shapley Values. Different colors represent the value of the feature from low (blue) to high (red).

6. Testing

You must have a simple test for each operator you implement and we strongly recommend using Pytest for this purpose. In case you are not familiar with Pytest, you might want to first spend some time reading the code snippets provided in the documentation.

All test functions must be added to the separate `tests.py` file provided with the code skeleton. Before submitting your solution, make sure the command `pytest tests.py` runs all your test functions successfully.

7. Logging

Logging can save you hours when debugging your program, and you will find it extremely helpful as your codebase grows in size and complexity. Always use Python's logger to generate messages and avoid using `print()` for that purpose. Keep in mind that logging has some performance overhead even if set to INFO level.

8. Git

You will use git to maintain your codebase and submit your solutions. If you are not familiar with git, you might want to have a look here. Note that well-documented code is always easier to understand and grade, so please make sure your code is clean, readable, and has comments.

Make sure your git commits have meaningful messages. Messages like "*Commit*" or "*Fix*", etc. are pretty vague and must be avoided. Always try to briefly describe what each commit is about, e.g. "*Add Join operator*", "*Fix provenance tracking in AVG operator*", etc., so that you can easily search your git log if necessary.

Each time you finish a task (including the related tests), we strongly recommend that you use a commit message "Complete *Task X*". You will find these commits very helpful in case you want to rollback and create new branches in your repository.

9. Deliverables

Each submission must be marked with a commit message "*Submit Assignment X*" in git. If there are multiple submissions with the same message, we will only consider the last one before the deadline. In case all your submission commits are after the deadline, we will only consider the last commit, however, **late submissions will be eligible for up to 50% of the original grade**.

Your submission must contain:

1. The code you wrote to solve the assignment tasks (in `assignment_3.py`)
2. The code you wrote for testing (in `tests.py`)
3. The explanations you generated for Task IV as images (in a new folder `images`)

Before submitting your solution, always make sure your code passes all tests successfully.

10. Resources

- Scikit-learn: https://scikit-learn.org/stable/
- InterpretML: https://interpret.ml
- LIME: https://github.com/marcotcr/lime

- SHAP: https://github.com/slundberg/shap
- Light Gradient Boosting Machine: https://lightgbm.readthedocs.io/en/latest/
- Python tutorial: https://docs.python.org/3/tutorial/
- Python logger: https://docs.python.org/3/library/logging.html
- Pytest: https://docs.pytest.org/en/stable/
- Ray documentation: https://docs.ray.io/en/latest/
- Git Handbook: https://guides.github.com/introduction/git-handbook/