EXPLAINABLE ARTIFICIAL INTELLIGENCE (PART 3)

Hands-on Machine Learning Model Interpretation

A comprehensive guide to interpreting machine learning models

Dipanjan (DJ) Sarkar Dec 14, 2018 · 26 min read

Introduction

Interpreting Machine Learning models is no longer a luxury but a necessity given the rapid adoption of AI in the industry. This article in a continuation in my series of articles aimed at ***'Explainable Artificial Intelligence (XAI)'***. The idea here is to cut through the hype and enable you with the tools and techniques needed to start interpreting any black box machine learning model. Following are the previous articles in the series in case you want to give them a quick skim (but are not mandatory for this article).

and importance of model interpretation along with its scope and criteria

- ***Part 2 —Model Interpretation Strategies'*** which covers the how of human interpretable machine learning where we look at essential concepts pertaining to major strategies for model interpretation.

In this article we will give you hands-on guides which showcase various ways to explain potential black-box machine learning models in a model-agnostic way. We will be working on a real-world dataset on Census income, also known as the *Adult dataset* available in the *UCI ML Repository* where we will be predicting if the potential income of people is more than $50K/yr or not.

The purpose of this article is manifold. The first main objective is to familiarize ourselves with the major state-of-the-art model interpretation frameworks out there (a lot of them being extensions of LIME — the original framework and approach proposed for model interpretation which we have covered in detail in *Part 2 of this series*).

We cover usage of the following model interpretation frameworks in our tutorial.

- **ELI5**
- **Skater**
- **SHAP**

The major model interpretation techniques we will be covering in this tutorial include the following.

- **Feature Importances**
- **Partial Dependence Plots**
- **Model Prediction Explanations with Local Interpretation**
- **Building Interpretable Models with Surrogate Tree-based Models**
- **Model Prediction Explanation with SHAP values**
- **Dependence & Interaction Plots with SHAP**

Loading Necessary Dependencies

We will be using a lot of frameworks and tools in this article given it is a hands-on guide to model interpretation. We recommend you to load up the following dependencies to get the maximum out of this guide!

```
In [1]: import pandas as pd
        import numpy as np
        import model_evaluation_utils as meu
        import matplotlib.pyplot as plt
        from collections import Counter
        import shap
        import eli5

        import warnings
        warnings.filterwarnings('ignore')
        plt.style.use('fivethirtyeight')
        %matplotlib inline

        shap.initjs()
```

Rememeber to call the `shap.initjs()` function since a lot of the plots from `shap` require JavaScript.

Load and View the Census Income Dataset

You can actually get the ***census income*** dataset (popularly known as the ***adult dataset***) from the *UCI ML repository*. Fortunately `shap` provides us an already cleaned up version of this dataset which we will be using here since the intent of this article is model interpretation.

Viewing the Data Attributes

Let's take a look at the major features or attributes of our dataset.

```
1  data, labels = shap.datasets.adult(display=True)
2  labels = np.array([int(label) for label in labels])
3
4  print(data.shape, labels.shape)
5  data.head()
```

xai_1.py hosted with ♡ by GitHub view raw

```
((32561, 12), (32561,))
```

1	50.0	Self-emp-not-inc	13.0	Married-civ-spouse	Exec-managerial	Husband	White	Male	0.0	0.0	13.0	United-States
2	38.0	Private	9.0	Divorced	Handlers-cleaners	Not-in-family	White	Male	0.0	0.0	40.0	United-States
3	53.0	Private	7.0	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0.0	0.0	40.0	United-States
4	28.0	Private	13.0	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0.0	0.0	40.0	Cuba

Census Dataset Features

We will explain these features shortly.

Viewing the Class Labels

Let's view the distribution of people with $<= \$50K$ (`False`) and $> \$50K$ (`True`) income which are our class labels which we want to predict.

```
In [6]: Counter(labels)

Out[6]: Counter({0: 24720, 1: 7841})
```

Definitely some class imbalance which is expected given that we should have less people having a higher income.

Understanding the Census Income Dataset

Let's now take a look at our dataset attributes and understand their meaning and significance.

Attribute Name	Type	Description
Age	Continuous	Represents age of the person
Workclass	Categorical	Represents the nature of working class\category (Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked)
Education-Num	Categorical	Numeric representation of educational qualification. Ranges from 1-16. (Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate

Marital Status	Categorical	Represents the marital status of the person (Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse)
Occupation	Categorical	Represents the type of profession\job of the person (Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces)
Relationship	Categorical	Represents the relationship status of the person (Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried)
Race	Categorical	Represents the race of the person (White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black)
Sex	Categorical	Represents the gender of the person (Female, Male)
Capital Gain	Continuous	The total capital gain for the person
Capital Loss	Continuous	The total capital loss for the person
Hours per week	Continuous	Total hours spent working per week
Country	Categorical	The country where the person is residing
Income Label (labels)	Categorical (class label)	The class label column is the one we want to predict (False: Income <= $50K & True: Income > $50K)

We have a total of 12 features and our objective is to predict if the income of a person will be **more than $50K** (`True`) or **less than $50K** (`False`). Hence we will be building and interpreting a classification model.

representations. Typically the XGBoost model can handle categorical data natively being a tree-based model so we don't one-hot encode the features here.

```
cat_cols = data.select_dtypes(['category']).columns
data[cat_cols] = data[cat_cols].apply(lambda x: x.cat.codes)
data.head()
```

xai_3.py hosted with ♡ by GitHub view raw

	Age	Workclass	Education-Num	Marital Status	Occupation	Relationship	Race	Sex	Capital Gain	Capital Loss	Hours per week	Country
0	39.0	7	13.0	4	1	1	4	1	2174.0	0.0	40.0	39
1	50.0	6	13.0	2	4	0	4	1	0.0	0.0	13.0	39
2	38.0	4	9.0	0	6	1	4	1	0.0	0.0	40.0	39
3	53.0	4	7.0	2	6	0	2	1	0.0	0.0	40.0	39
4	28.0	4	13.0	2	10	5	2	0	0.0	0.0	40.0	5

Time to build our train and test datasets before we build our classification model.

Building Train and Test Datasets

For any machine learning model, we always need *train* and *test* datasets. We will be building the model on the *train dataset* and test the performance on the *test dataset*. We maintain two datasets (one with the encoded categorical values and one with the original values) so we can train with the encoded dataset but use the original dataset as needed later on for model interpretation.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.3, random_state=42
print(X_train.shape, X_test.shape)
X_train.head(3)
```

xai_4.py hosted with ♡ by GitHub view raw

```
((22792, 12), (9769, 12))
```

	Age	Workclass	Education-Num	Marital Status	Occupation	Relationship	Race	Sex	Capital Gain	Capital Loss	Hours per week	Country
19749	34.0	6	9.0	2	5	5	4	0	0.0	2179.0	12.0	39
1216	48.0	6	10.0	2	3	0	0	1	7688.0	0.0	40.0	39

We also maintain our base dataset with the actual (not encoded) values also in a separate dataframe (useful for model interpretation later).

```
data_disp, labels_disp = shap.datasets.adult(display=True)
X_train_disp, X_test_disp, y_train_disp, y_test_disp = train_test_split(data_disp, labels_disp,
print(X_train_disp.shape, X_test_disp.shape)
X_train_disp.head(3)
```

xai_5.py hosted with ♡ by GitHub view raw

```
((22792, 12), (9769, 12))
```

	Age	Workclass	Education-Num	Marital Status	Occupation	Relationship	Race	Sex	Capital Gain	Capital Loss	Hours per week	Country
19749	34.0	Self-emp-not-inc	9.0	Married-civ-spouse	Farming-fishing	Wife	White	Female	0.0	2179.0	12.0	United-States
1216	48.0	Self-emp-not-inc	10.0	Married-civ-spouse	Craft-repair	Husband	Amer-Indian-Eskimo	Male	7688.0	0.0	40.0	United-States
27962	23.0	State-gov	10.0	Married-civ-spouse	Prof-specialty	Husband	White	Male	0.0	0.0	30.0	United-States

Our actual dataset

Training the classification model

We will now train and build a basic boosting classification model on our training data using the popular XGBoost framework, an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.

```
%%time

import xgboost as xgb
xgc = xgb.XGBClassifier(n_estimators=500, max_depth=5, base_score=0.5,
                        objective='binary:logistic', random_state=42)
xgc.fit(X_train, y_train)
```

xai_6.py hosted with ♡ by GitHub view raw

```
Wall time: 8.16 s
```

```
predictions = xgc.predict(X_test)
predictions[:10]
```

xai_7.py hosted with ♡ by GitHub view raw

```
array([0, 0, 1, 0, 0, 1, 1, 0, 0, 1])
```

Model Performance Evaluation

Time to put the model to the test! Let's evaluate how our model has performed with its predictions on the test data. We use my nifty `model_evaluation_utils` module for this which leverages `scikit-learn` internally to give us standard classification model evaluation metrics.

```
class_labels = list(set(labels))
meu.display_model_performance_metrics(true_labels=y_test,
                                      predicted_labels=predictions,
                                      classes=class_labels)
```

xai_8.py hosted with ♡ by GitHub view raw

```
Model Performance metrics:
------------------------------
Accuracy: 0.8712
Precision: 0.8671
Recall: 0.8712
F1 Score: 0.8681

Model Classification report:
------------------------------
              precision    recall  f1-score   support

           0       0.90      0.94      0.92      7455
           1       0.76      0.67      0.71      2314

   micro avg       0.87      0.87      0.87      9769
   macro avg       0.83      0.80      0.81      9769
weighted avg       0.87      0.87      0.87      9769

Prediction Confusion Matrix:
------------------------------
                Predicted:
                     0     1
Actual: 0         6972   483
        1          775  1539
```

Default Model Interpretation Methods

By default it is difficult to gauge on specific model interpretation methods for machine learning models out of the box. Parametric models like logistic regression are easier to interpret given that the total number of parameters of the model are fixed regardless of the volume of data and one can make some interpretation of the model's prediction decisions leveraging the parameter coefficients.

Some non-parametric models like tree-based models do have some out of the box model interpretation methods like feature importance which helps us in understanding which features might be influential in the model making its prediction decisions.

Classic feature importances from XGBoost

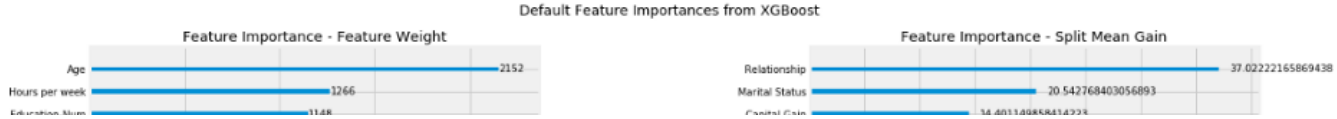
Here we try out the global feature importance calcuations that come with XGBoost. The model enables us to view feature importances based on the following.

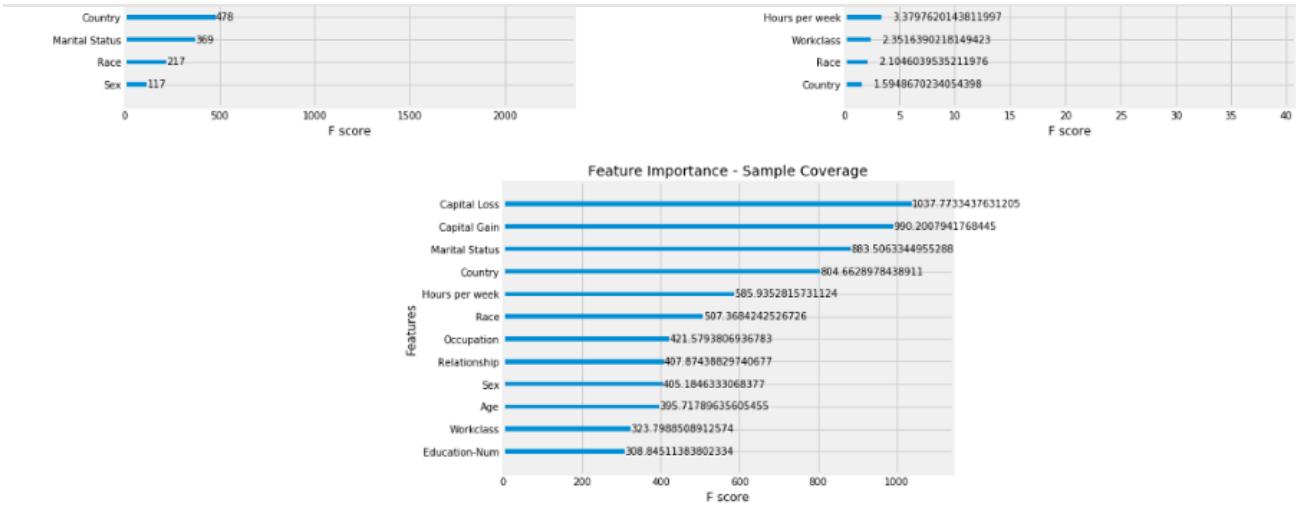
- **Feature Weights:** This is based on the number of times a feature appears in a tree across the ensemble of trees
- **Gain:** This is based on the average gain of splits which use the feature
- **Coverage:** This is based on the average coverage (number of samples affected) of splits which use the feature

Note that they all contradict each other, which motivates the use of model interpretation frameworks like SHAP which uses something known as SHAP values, which claim to come with consistency guarantees (meaning they will typically order the features correctly).

```
fig = plt.figure(figsize = (16, 12))
title = fig.suptitle("Default Feature Importances from XGBoost", fontsize=14)

ax1 = fig.add_subplot(2,2, 1)
xgb.plot_importance(xgc, importance_type='weight', ax=ax1)
t=ax1.set_title("Feature Importance - Feature Weight")

ax2 = fig.add_subplot(2,2, 2)
xgb.plot_importance(xgc, importance_type='gain', ax=ax2)
t=ax2.set_title("Feature Importance - Split Mean Gain")

ax3 = fig.add_subplot(2,2, 3)
xgb.plot_importance(xgc, importance_type='cover', ax=ax3)
t=ax3.set_title("Feature Importance - Sample Coverage")
```

xai_9.py hosted with ♡ by GitHub view raw





Feature Importance Plots from XGBoost

Model Interpretation with ELI5

ELI5 is a Python package which helps to debug machine learning classifiers and explain their predictions in an easy to understand an intuitive way. It is perhaps the easiest of the three machine learning frameworks to get started with since it involves minimal reading of documentation! However it doesn't support true model-agnostic interpretations and support for models are mostly limited to tree-based and other parametric\linear models. Let's look at some intuitive ways of model interpretation with ELI5 on our classification model.

Installation Instructions

We recommend installing this framework using `pip install eli5` since the `conda` version appears to be a bit out-dated. Also feel free to check out **the documentation** as needed.

Feature Importances with ELI5

Typically for tree-based models ELI5 does nothing special but uses the out-of-the-box feature importance computation methods which we discussed in the previous section. By default, **'gain'** is used, that is the average gain of the feature when it is used in trees.

```
eli5.show_weights(xgc.get_booster())
```

Weight	Feature
0.3294	Relationship

0.0494	Sex
0.0368	Occupation
0.0327	Age
0.0301	Hours per week
0.0209	Workclass
0.0187	Race
0.0142	Country

Explaining Model Prediction Decisions with ELI5

One of the best way to explain model prediction decisions to either a technical or a more business-oriented individual, is to examine individual data-point predictions. Typically, ELI5 does this by showing weights for each feature depicting how influential it might have been in contributing to the final prediction decision across all trees. The idea for weight calculation is described **here**; ELI5 provides an independent implementation of this algorithm for XGBoost and most scikit-learn tree ensembles which is definitely on the path towards model-agnostic interpretation but not purely model-agnostic like LIME.

Typically, the prediction can be defined as the sum of the feature contributions + the "bias" (i.e. the mean given by the topmost region that covers the entire training set)

Predicting when a person's income <= $50K

Here we can see the most influential features being the **`Age`, `Hours per week`, `Marital Status`, `Occupation` & `Relationship`**

```
doc_num = 0
print('Actual Label:', y_test[doc_num])
print('Predicted Label:', predictions[doc_num])
eli5.show_prediction(xgc.get_booster(), X_test.iloc[doc_num],
                     feature_names=list(data.columns),
                     show_feature_values=True)
```

xai_10.py hosted with ♡ by GitHub view raw

```
Actual Label: 0
Predicted Label: 0
```

Out[16]: **y (score -6.069) top features**

Contribution[?]	Feature	Value
+0.009	Race	4.000
-0.004	Education-Num	10.000
-0.004	Country	39.000
-0.006	Workclass	4.000

-0.639	Occupation	1.000
-0.731	Marital Status	0.000
-0.839	Hours per week	38.000
-1.363	<BIAS>	1.000
-1.470	Age	27.000

Predicting when a person's income > $50K

Here we can see the most influential features being the **`Education`**, **`Relationship`**, **`Occupation`**, **`Hours per week`** & **`Marital Status`**

```
doc_num = 2
print('Actual Label:', y_test[doc_num])
print('Predicted Label:', predictions[doc_num])
eli5.show_prediction(xgc.get_booster(), X_test.iloc[doc_num],
                     feature_names=list(data.columns),
                     show_feature_values=True)
```

xai_11.py hosted with ♡ by GitHub view raw

```
Actual Label: 1
Predicted Label: 1
```

Out[17]: **y (score 0.824) top features**

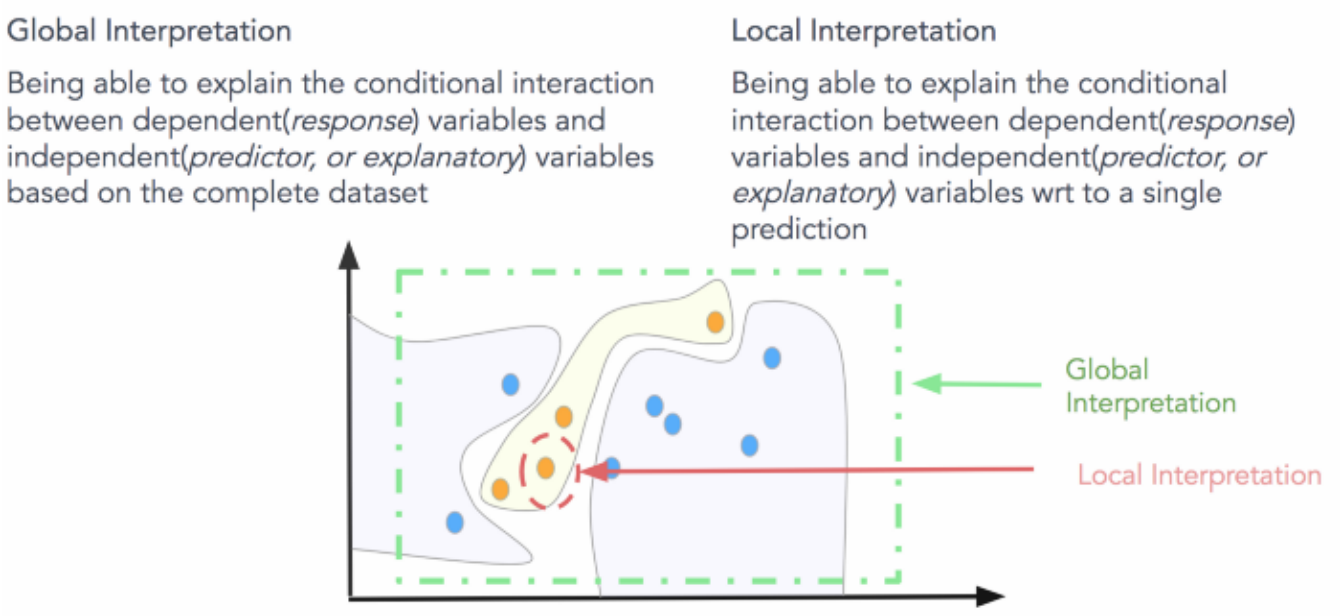
Contribution[?]	Feature	Value
+1.266	Education-Num	13.000
+0.532	Relationship	0.000
+0.514	Occupation	4.000
+0.496	Hours per week	55.000
+0.468	Marital Status	2.000
+0.057	Workclass	4.000
+0.035	Sex	1.000
+0.018	Country	39.000
-0.120	Capital Loss	0.000
-0.203	Capital Gain	0.000
-0.279	Race	2.000
-0.598	Age	29.000
-1.363	<BIAS>	1.000

It is definitely interesting to see how similar features play an influential role in explaining model prediction decisions for both classes!

Model Interpretation with Skater

Skater is a unified framework to enable Model Interpretation for all forms of models to help one build an Interpretable machine learning system often needed for real world use-cases using a model-agnostic approach. It is an open source python library

individual prediction).



Skater originally started off as a fork of LIME but then broke out as an independent framework of it's own with a wide variety of feature and capabilities for model-agnostic interpretation for any black-box models. The project was started as a research idea to find ways to enable better interpretability(preferably human interpretability) to predictive *"black boxes"* both for researchers and practitioners.

Install Skater

You can typically install Skater using a simple `pip install skater`. For detailed information on the dependencies and installation instruction check out installing skater.

📖 Documentation

We recommend you to check out the detailed documentation of Skater.

Overview	Introduction to the Skater library
Installing	How to install the Skater library
Tutorial	Steps to use Skater effectively.
API Reference	The detailed reference for Skater's API.
Contributing	Guide to contributing to the Skater project.

Algorithms

Skater has a suite of model interpretation techniques some of which are mentioned below.

```
+---------+---------+-----+-----------+-----------+--------------+--------------+--------------------+------------------+
| Scope of Interpretation |           Algorithms                                                                          |
+=========+=========+=====+===========+===========+==============+==============+====================================+
| Global Interpretation   | `Model agnostic Feature Importance <https://tinyurl.com/feature-importance>`_                 |
+---------+---------+-----+-----------+-----------+--------------+--------------+--------------------+------------------+
| Global Interpretation   | `Model agnostic Partial Dependence Plots <https://tinyurl.com/partial-dependence>`_           |
+---------+---------+-----+-----------+-----------+--------------+--------------+--------------------+------------------+
| Local Interpretation    | `Local Interpretable Model Explanation(LIME) <https://tinyurl.com/lime-explanation>`_         |
+---------+---------+-----+-----------+-----------------------------------------+--------------------+------------------+
| Local Interpretation    | DNNs      | - `Layer-wise Relevance Propagation <https://tinyurl.com/e-layerwise>`_           |
|                         |           |   (e-LRP): image                                                                  |
|                         |           |                                                                                   |
|                         |           | - `Occlusion  <https://tinyurl.com/dnn-occlusion>`_ : image                       |
|                         |           |                                                                                   |
|                         |           | - `Integrated Gradient <https://tinyurl.com/integrated-gradient>`_                |
|                         |           |   image and text                                                                  |
+---------+---------+-----+-----------+-----------------------------------------+--------------------+------------------+
| Global and Local        | `Scalable Bayesian Rule Lists <https://tinyurl.com/rule-list-sbr>`_                           |
| Interpretation          |                                                                                               |
|                         | `Tree Surrogates <https://tinyurl.com/treesurrogates>`_                                       |
+---------+---------+-----+-----------+-----------+--------------+--------------+--------------------+------------------+
```

Usage and Examples

Since the project is under active development, the best way to understand usage would be to follow the examples mentioned in the Gallery of Interactive Notebook. But we will be showcasing its major capabilities using the model trained on our census dataset.

Global Interpretations with Skater

A predictive model is a mapping from an input space to an output space. Interpretation algorithms are divided into those that offer statistics and metrics on regions of the domain, such as the marginal distribution of a feature, or the joint distribution of the entire training set. In an ideal world there would exist some representation that would allow a human to interpret a decision function in any number of dimensions. Given that we generally can only intuit visualizations of a few dimensions at time, global interpretation algorithms either aggregate or subset the feature space.

Currently, model-agnostic global interpretation algorithms supported by skater include partial dependence and feature importance with a very new release of tree-surrogates also. We will be covering feature importance and partial dependence plots here.

model, and run interpretation algorithms. Typically, an `Interpretation` consumes a dataset, and optionally some metadata like feature names and row ids. Internally, the `Interpretation` will generate a `DataManager` to handle data requests and sampling.

- **Local Models(`InMemoryModel`):** To create a skater model based on a local function or method, pass in the predict function to an `InMemoryModel` . A user can optionally pass data samples to the examples keyword argument. This is only used to infer output types and formats. Out of the box, skater allows models return `numpy` arrays and `pandas` dataframes.
- **Operationalized Model(`DeployedModel`):** If your model is accessible through an API, use a `DeployedModel` , which wraps the requests library. `DeployedModels` require two functions, an input formatter and an output formatter, which speak to the requests library for posting and parsing. The input formatter takes a `pandas` DataFrame or a `numpy` ndarray, and returns an object (such as a dict) that can be converted to JSON to be posted. The output formatter takes a requests.response as an input and returns a `numpy` ndarray or `pandas` DataFrame.

We will use the following workflow:

- Build an interpretation object
- Build an in-memory model
- Perform interpretations

```
from skater.core.explanations import Interpretation
from skater.model import InMemoryModel

interpreter = Interpretation(training_data=X_test, training_labels=y_test,
                             feature_names=list(data.columns))
im_model = InMemoryModel(xgc.predict_proba, examples=X_train,
                         target_names=['$50K or less', 'More than $50K'])
```

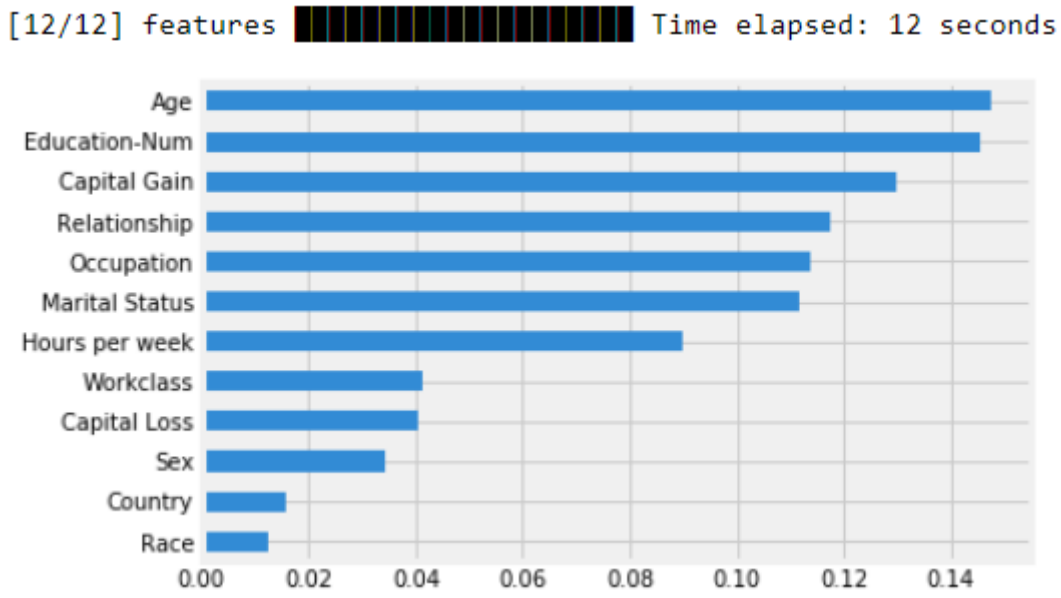
xai_13.py hosted with ♡ by GitHub view raw

Feature Importances with Skater

information theoretic criteria, measuring the entropy in the change of predictions, given a perturbation of a given feature. The intuition is that the more a model's decision criteria depend on a feature, the more we'll see predictions change as a function of perturbing a feature. The default method used is `prediction-variance` which is the mean absolute value of changes in predictions, given perturbations in the data.

```
plots = interpreter.feature_importance.plot_feature_importance(im_model, ascending=True,
                                                               n_samples=23000)
```

xai_14.py hosted with ♡ by GitHub view raw



Feature Importances from Skater

Partial Dependence

Partial Dependence describes the marginal impact of a feature on model prediction, holding other features in the model constant. The derivative of partial dependence describes the impact of a feature (analogous to a feature coefficient in a regression model). This has been adapted from *T. Hastie, R. Tibshirani and J. Friedman, Elements of Statistical Learning Ed. 2, Springer*. 2009.

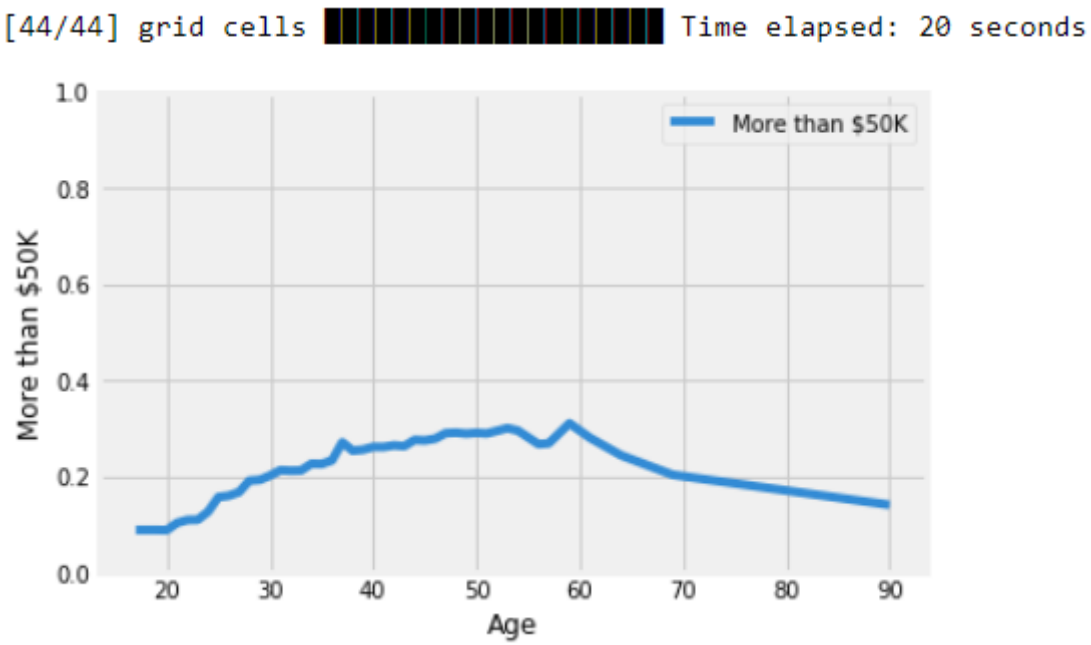
The partial dependence plot (PDP or PD plot) shows the marginal effect of a feature on the predicted outcome of a previously fit model. PDPs can show if the relationship

PDP of 'Age' affecting model prediction

Let's take a look at how the `Age` feature affects model predictions.

```
r = interpreter.partial_dependence.plot_partial_dependence(['Age'], im_model, grid_resolution=50
                                                           grid_range=(0,1), n_samples=23000,
                                                           with_variance=True, figsize = (6, 4))
yl = r[0][1].set_ylim(0, 1)
```

xai_15.py hosted with ♡ by GitHub view raw



PDP for the Age feature

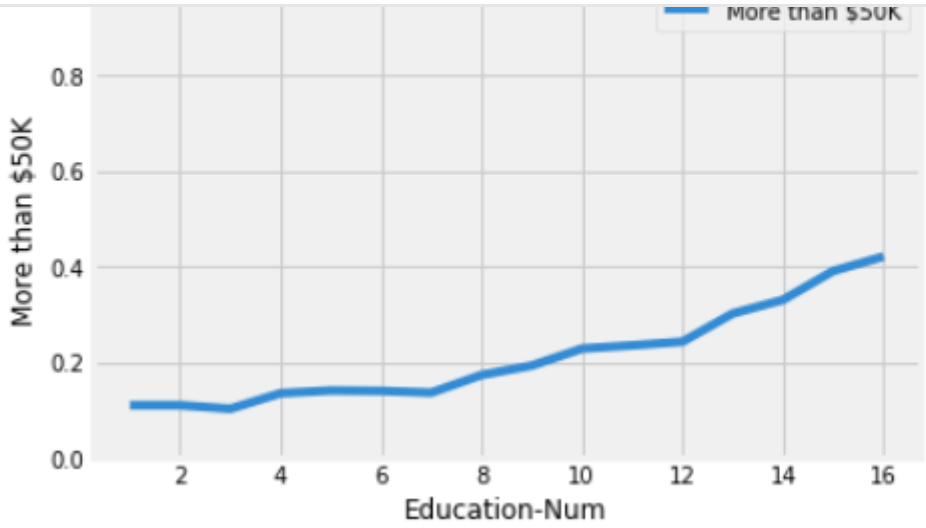
Looks like the middle-aged people have a slightly higher chance of making more money as compared to younger or older people.

PDP of 'Education Num' affecting model prediction

Let's take a look at how the `Education-Num` feature affects model predictions.

```
r = interpreter.partial_dependence.plot_partial_dependence(['Education-Num'], im_model, grid_res
                                                           grid_range=(0,1), n_samples=23000,
                                                           with_variance=True, figsize = (6, 4))
yl = r[0][1].set_ylim(0, 1)
```

xai_16.py hosted with ♡ by GitHub view raw



PDP for the Education-Num feature

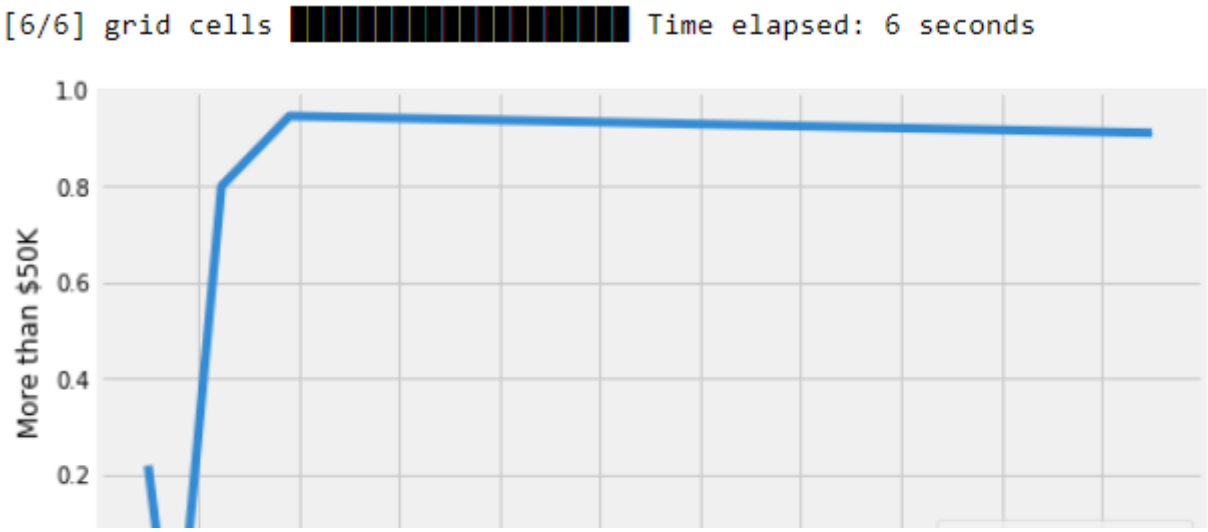
Looks like higher the education level, the better the chance of making more money. Not surprising!

PDP of 'Capital Gain' affecting model prediction

Let's take a look at how the `Capital Gain` feature affects model predictions.

```
r = interpreter.partial_dependence.plot_partial_dependence(['Capital Gain'], im_model, grid_resc
                                                           grid_range=(0,1),
                                                           with_variance=True, figsize = (8, 4),
yl = r[0][1].set_ylim(0, 1)
s, e = r[0][1].get_xlim()
xl = r[0][1].set_xticks(np.arange(s, e, 10000))
```

xai_17.py hosted with ♡ by GitHub view raw



PDP for the Capital Gain feature

Unsurprisingly higher the capital gain, the more chance of making money, there is a steep rise in around **$5K — $8K.**

PDP of 'Relationship' affecting model prediction

Remember that relationship is coded as a categorical variable with numeric representations. Let's first look at how it is represented.

```
pd.concat([data_disp[['Relationship']], data[['Relationship']]],
          axis=1).drop_duplicates()
```

xai_18.py hosted with ♡ by GitHub view raw

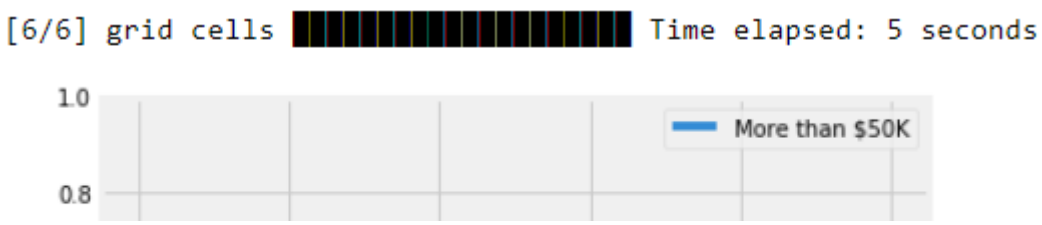
	Relationship	Relationship
0	Not-in-family	1
1	Husband	0
4	Wife	5
12	Own-child	3
17	Unmarried	4
74	Other-relative	2

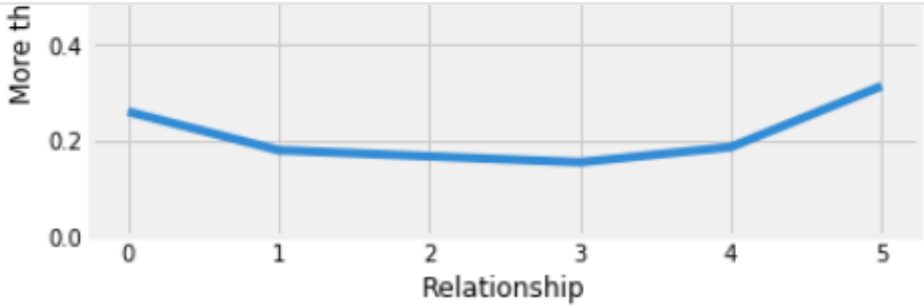
Label Encoding for Relationship

Let's now take a look at how the `Relationship` feature affects model predictions.

```
r = interpreter.partial_dependence.plot_partial_dependence(['Relationship'], im_model, grid_resc
                                                           grid_range=(0,1), n_samples=23000,
                                                           with_variance=True, figsize = (6, 4))
yl = r[0][1].set_ylim(0, 1)
```

xai_19.py hosted with ♡ by GitHub view raw





PDP for the Relationship feature

Interesting definitely that married folks (husband-wife) have a higher chance of making more money than others!

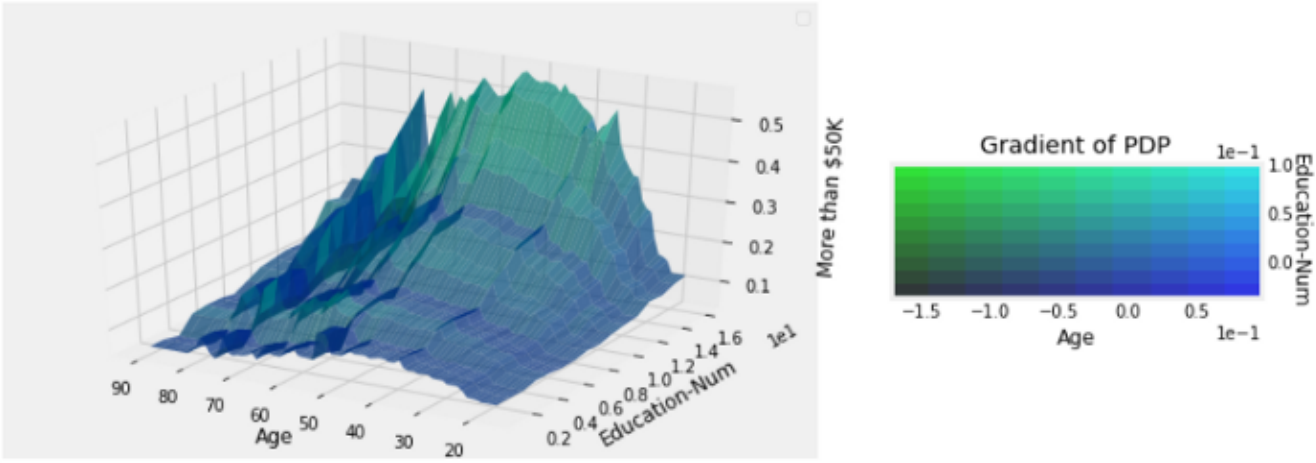
Two-way PDP showing interactions between features 'Age' and 'Education-Num' and their effect on making more than $50K

We run a deeper model interpretation here over all the data samples, trying to see interactions between `Age` and `Education-Num` and also their effect on the probability of the model predicting if the person will make more money, with the help of a two-way partial dependence plot.

```
plots_list = interpreter.partial_dependence.plot_partial_dependence([('Age', 'Education-Num')],
                                                                    im_model, grid_range=(0,1),
                                                                    n_samples=23000,
                                                                    figsize=(12, 5),
                                                                    grid_resolution=100)
```

xai_20.py hosted with ♡ by GitHub view raw

```
[1136/1136] grid cells ████████████████████ Time elapsed: 629 seconds
```



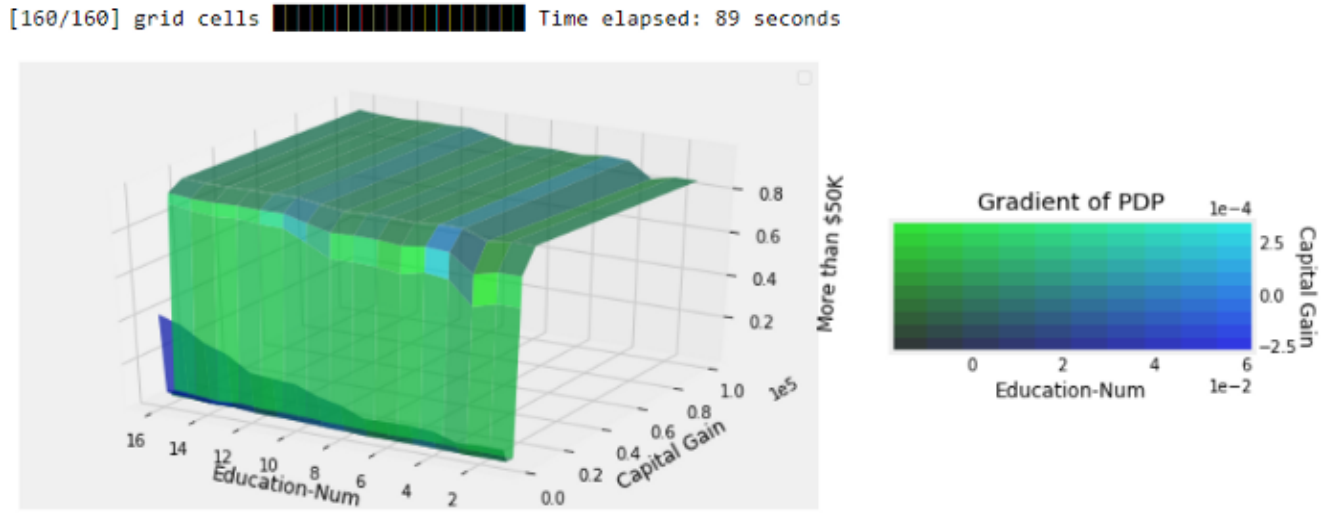
interesting to see higher the education level and the middle-aged folks (30–50) having the highest chance of making more money!

Two-way PDP showing interactions between features 'Education-Num' and 'Capital Gain' and their effect on making more than $50K

We run a deeper model interpretation here over all the data samples, trying to see interactions between `Education-Num` and `Capital Gain` and also their effect on the probability of the model predicting if the person will make more money, with the help of a two-way partial dependence plot.

```
plots_list = interpreter.partial_dependence.plot_partial_dependence([('Education-Num', 'Capital
                                                                      im_model, grid_range=(0,1),
                                                                      n_samples=23000,
                                                                      figsize=(12, 5),
                                                                      grid_resolution=100)
```

xai_21.py hosted with ♡ by GitHub view raw



Two-way PDP showing effects of the Education-Num and Capital Gain features

Basically having a better education and more capital gain leads to you making more money!

Local Interpretations with Skater

Local Interpretation could be possibly be achieved in two ways. Firstly, one could possibly approximate the behavior of a complex predictive model in the vicinity of a single input using a simple interpretable auxiliary or surrogate model (e.g. Linear

Local Interpretable Model-Agnostic Explanations(LIME)

LIME is a novel algorithm designed by Riberio Marco, Singh Sameer, Guestrin Carlos to access the behavior of the any base estimator(model) using interpretable surrogate models (e.g. linear classifier/regressor). Such form of comprehensive evaluation helps in generating explanations which are locally faithful but may not align with the global behavior. Basically, LIME explanations are based on local surrogate models. These, surrogate models are interpretable models (like a linear model or decision tree) that are learned on the predictions of the original black box model. But instead of trying to fit a global surrogate model, LIME focuses on fitting local surrogate models to explain why single predictions were made.

The idea is very intuitive. To start with, just try and unlearn what you have done so far! Forget about the training data, forget about how your model works! Think that your model is a black box model with some magic happening inside, where you can input data points and get the models predicted outcomes. You can probe this magic black box as often as you want with inputs and get output predictions.

Now, you main objective is to understand why the machine learning model which you are treating as a magic black box, gave the outcome it produced. LIME tries to do this for you! It tests out what happens to you black box model's predictions when you feed variations or perturbations of your dataset into the black box model. Typically, LIME generates a new dataset consisting of perturbed samples and the associated black box model's predictions. On this dataset LIME then trains an interpretable model weighted by the proximity of the sampled instances to the instance of interest. Following is a standard high-level workflow for this.

- Choose your instance of interest for which you want to have an explanation of the predictions of your black box model.
- Perturb your dataset and get the black box predictions for these new points.
- Weight the new samples by their proximity to the instance of interest.
- Fit a weighted, interpretable (surrogate) model on the dataset with the variations.
- Explain prediction by interpreting the local model.

Explaining Model Predictions with Skater using LIME

Skater can leverage LIME to explain model predictions. Typically, its `LimeTabularExplainer` class helps in explaining predictions on tabular (i.e. matrix) data. For numerical features, it perturbs them by sampling from a Normal(0,1) and doing the inverse operation of mean-centering and scaling, according to the means and stds in the training data. For categorical features, it perturbs by sampling according to the training distribution, and making a binary feature that is 1 when the value is the same as the instance being explained. The `explain_instance()` function generates explanations for a prediction. First, we generate neighborhood data by randomly perturbing features from the instance. We then learn locally weighted linear (surrogate) models on this neighborhood data to explain each of the classes in an interpretable way.

Since XGBoost has some issues with feature name ordering when building models with dataframes, we will build our same model with `numpy` arrays to make LIME work without additional hassles of feature re-ordering. Remember the model being built is the same ensemble model which we treat as our black box machine learning model.

```
xgc_np = xgb.XGBClassifier(n_estimators=500, max_depth=5, base_score=0.5,
                           objective='binary:logistic', random_state=42)
xgc_np.fit(X_train.values, y_train)
```

xai_22.py hosted with ♡ by GitHub view raw

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bytree=1, gamma=0, learning_rate=0.1,
              max_delta_step=0, max_depth=5, min_child_weight=1,
              missing=None, n_estimators=500, n_jobs=1,
              nthread=None, objective='binary:logistic',
              random_state=42, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, seed=None, silent=True,
              subsample=1)
```

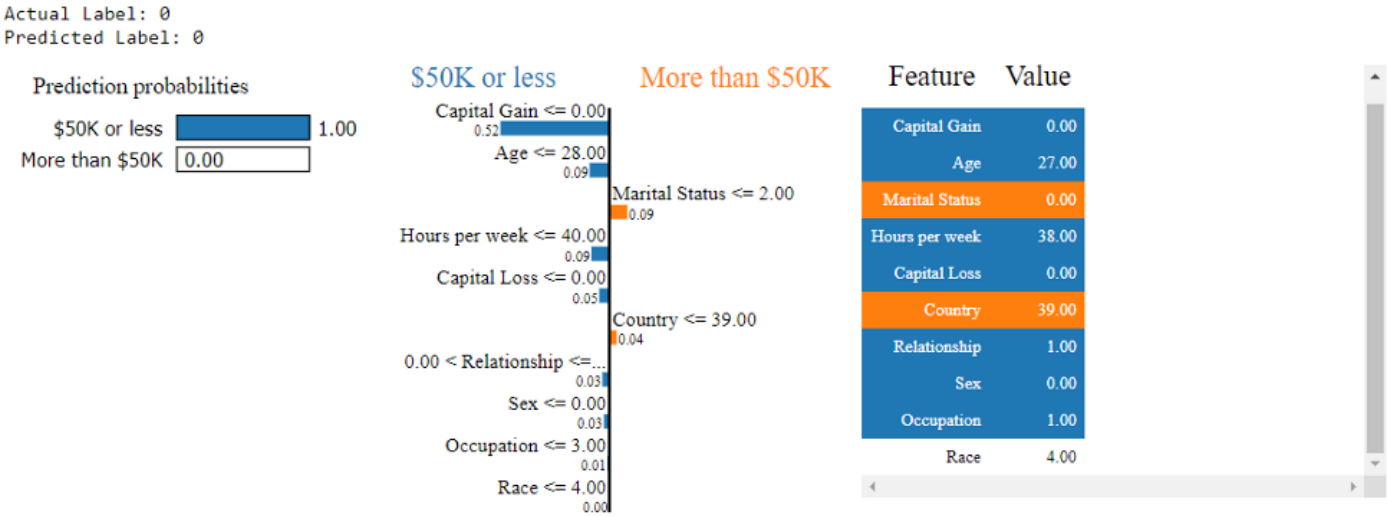
```
from skater.core.local_interpretation.lime.lime_tabular import LimeTabularExplainer

exp = LimeTabularExplainer(X_test.values, feature_names=list(data.columns),
                           discretize continuous=True,
```

Predicting when a person's income <= $50K

Skater gives a nice reasoning below showing which features were the most influential in the model taking the correct decision of predicting the person's income as ***below $50K.***

```
doc_num = 0
print('Actual Label:', y_test[doc_num])
print('Predicted Label:', predictions[doc_num])
exp.explain_instance(X_test.iloc[doc_num].values, xgc_np.predict_proba).show_in_notebook()
```

xai_24.py hosted with ♡ by GitHub view raw



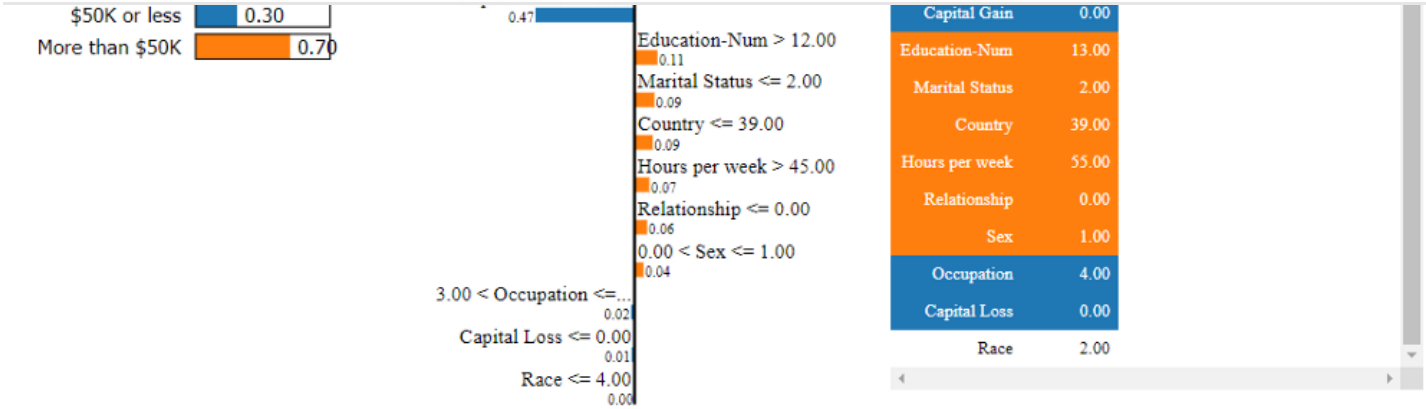
Local interpretations with LIME in Skater

Predicting when a person's income > $50K

Skater gives a nice reasoning below showing which features were the most influential in the model taking the correct decision of predicting the person's income as ***above $50K.***

```
doc_num = 2
print('Actual Label:', y_test[doc_num])
print('Predicted Label:', predictions[doc_num])
exp.explain_instance(X_test.iloc[doc_num].values, xgc_np.predict_proba).show_in_notebook()
```

xai_25.py hosted with ♡ by GitHub view raw



Local interpretations with LIME in Skater

Path to more interpretable models with Tree Surrogates using Skater

We have see various ways to interpret machine learning models with features, dependence plots and even LIME. But can we build an approximation or a surrogate model which is more interpretable from a really complex black box model like our XGBoost model having hundreds of decision trees?

Here in, we introduce the novel idea of using `TreeSurrogates` as means for explaining a model's learned decision policies (for inductive learning tasks), which is inspired by the work of Mark W. Craven described as the TREPAN algorithm.

We recommend checking out the following excellent papers on the TREPAN algorithm to build surrogate trees.

- *Mark W. Craven(1996) EXTRACTING COMPREHENSIBLE MODELS FROM TRAINED NEURAL NETWORKS*
- *Mark W. Craven and Jude W. Shavlik(NIPS, 96). Extracting Thee-Structured Representations of Thained Networks*

Briefly, Trepan constructs a decision tree in a best-first manner. It maintains a queue of leaves which are expanded into subtrees as they are removed from the queue. With each node in the queue, Trepan stores,

- A subset of the training examples,
- Another set of instances (query instances),
- A set of constraints.

splitting test if the node is an internal node or to determine the class label if it is a leaf. The constraint set describes the conditions that instances must satisfy in order to reach the node; this information is used when drawing a set of query instances for a newly created node. The process of expanding a node in Trepan is much like it is in conventional decision tree algorithms: a splitting test is selected for the node, and a child is created for each outcome of the test. Each child is either made a leaf of the tree or put into the queue for future expansion.

For Skater's implementation, for building explainable surrogate models, the base estimator(Oracle) could be any form of a supervised learning predictive model — our black box model. The explanations are approximated using Decision Trees(both for Classification/Regression) by learning decision boundaries similar to that learned by the Oracle (predictions from the base model are used for learning the Decision Tree representation). The implementation also generates a fidelity score to quantify tree based surrogate model's approximation to the Oracle. Ideally, the score should be 0 for truthful explanation both globally and locally. Let's check this out in action!

> ***NOTE:*** *The implementation is currently experimental and might change in future.*

Using the interpreter instance invoke call to the TreeSurrogate

We can use the Interpretation object we instantiated earlier to invoke a call to the TreeSurrogate capability.

Using the surrogate model to learn the decision boundaries learned by the base estimator

We can now fit this surrogate model on our dataset to learn the decision boundaries of our base estimator.

- Reports the fidelity value when compared to the base estimator (closer to 0 is better)
- Learner uses F1-score as the default metric of choice for classification.

```
0.009
```

Taking a look at the position for each feature

Visualizing the Surrogate Tree

We can now visualize our surrogate tree model using the following code.

Our surrogate tree model

Interesting rules from the surrogate tree

Here are some interesting rules you can observe from the above tree.

- If `Relationship` < 0.5 (means 0) and `Education-num` <= 9.5 and `Capital Gain` <= 4225 → ***70% chance of person making <= $50K***
- If `Relationship` < 0.5 (means 0) and `Education-num` <= 9.5 and `Capital Gain` >= 4225 → ***94.5% chance of person making > $50K***
- If `Relationship` < 0.5 (means 0) and `Education-num` >= 9.5 and `Education-num` is also >= 12.5 → ***94.7% chance of person making > $50K***

Surrogate Model Performance Evaluation

Let's check the performance of our surrogate model now on the test data.

Just as expected, the model performance drops a fair bit but still we get an overall ***F1-score*** of **83%** as compared to our boosted model's score of **87%** which is quite good!

Model Interpretation with SHAP

SHAP (SHapley Additive exPlanations) is a unified approach to explain the output of any machine learning model. SHAP connects game theory with local explanations, uniting several previous methods and representing the only possible consistent and locally accurate additive feature attribution method based on what they claim! (do check out the SHAP NIPS paper for details). We have also covered this in detail in *Part 2* of this series.

Installation

SHAP can be installed from PyPI

```
pip install shap
```

or conda-forge

```
conda install -c conda-forge shap
```

The really awesome aspect about this framework is while SHAP values can explain the output of any machine learning model, for really complex ensemble models it can be

LightGBM, *CatBoost*, and *scikit-learn* tree models!

SHAP (SHapley Additive exPlanations) assigns each feature an importance value for a particular prediction. Its novel components include: the identification of a new class of additive feature importance measures, and theoretical results showing there is a unique solution in this class with a set of desirable properties. Typically, SHAP values try to explain the output of a model (function) as a sum of the effects of each feature being introduced into a conditional expectation. Importantly, for non-linear functions the order in which features are introduced matters. The SHAP values result from averaging over all possible orderings. Proofs from game theory show this is the only possible consistent approach.

An intuitive way to understand the Shapley value is the following: The feature values enter a room in random order. All feature values in the room participate in the game (= contribute to the prediction). The Shapley value $\boldsymbol{\phi_{ij}}$ is the average marginal contribution of feature value $\boldsymbol{x_{ij}}$ by joining whatever features already entered the room before, i.e.

The following figure from the KDD 18 paper, *Consistent Individualized Feature Attribution for Tree Ensembles* summarizes this in a nice way!

Let's now dive into SHAP and leverage it for interpreting our model!

Explain predictions with SHAP

models you are building. We estimate the SHAP values for a set of samples (test data)

```
Expected Value: -1.3625857
```


This returns a matrix of SHAP values (`# samples , # features`). Each row sums to the difference between the model output for that sample and the expected value of the model output (which is stored as `expected_value` attribute of the explainer). Typically this difference helps us in explaining why the model is inclined on predicting a specific class outcome.

Predicting when a person's income <= $50K

SHAP gives a nice reasoning below showing which features were the most influential in the model taking the correct decision of predicting the person's income as ***below $50K***. The below explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the actual model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in blue.

Predicting when a person's income > $50K

Similarly, SHAP gives a nice reasoning below showing which features were the most influential in the model taking the correct decision of predicting the person's income as ***greater than $50K***.

Visualizing and explaining multiple predictions

One of the key advantages of SHAP is it can build beautiful interactive plots which can visualize and explain multiple predictions at once. Here we visualize model prediction decisions for the first 1000 test data samples.

The above visualization can be interacted with in multiple ways. The default visualization shows some interesting model prediction pattern decisions.

- The first 100 test samples all probably **earn more than $50K** and they **are married** or\and have a **good capital gain** or\and have a **higher education level**!
- The next 170+ test samples all probably **earn less than or equal to $50K** and they **are mostly un-married** and\or are **very young in age or divorced**!
- The next 310+ test samples have an inclination towards mostly **earning more than $50K** and they are of diverse profiles including married folks, people with different age and education levels and occupation. Most dominant features pushing the model towards making a prediction for higher income is the person being married i.e. **relationship: husband or wife**!

relationship: either unmarried or divorced and **very young in age**!

Definitely interesting how we can find out patterns which lead to the model making specific decisions and being able to provide explanations for them.

Feature Importances with SHAP

This basically takes the average of the SHAP value magnitudes across the dataset and plots it as a simple bar chart.

SHAP feature importance plot

SHAP Summary Plot

Besides a typical feature importance bar chart, SHAP also enables us to use a density scatter plot of SHAP values for each feature to identify how much impact each feature has on the model output for individuals in the validation dataset. Features are sorted by the sum of the SHAP value magnitudes across all samples. Note that when the scatter points don't fit on a line they pile up to show density, and the color of each point represents the feature value of that individual.

SHAP summary plot

It is interesting to note that the ***age*** and ***marital status*** feature has more total model impact than the ***capital gain*** feature, but for those samples where ***capital gain*** matters it has more impact than ***age*** or ***marital status***. In other words, ***capital gain*** affects a few predictions by a large amount, while ***age*** or ***marital status*** affects all predictions by a smaller amount.

SHAP Dependence Plots

SHAP dependence plots show the effect of a single (or two) feature across the whole dataset. They plot a feature's value vs. the SHAP value of that feature across many samples. SHAP dependence plots are similar to partial dependence plots, but account for the interaction effects present in the features, and are only defined in regions of the input space supported by data. The vertical dispersion of SHAP values at a single feature value is driven by interaction effects, and another feature can be chosen for coloring to highlight possible interactions.

You will also notice its similarity with Skater's Partial Dependence Plots!

PDP of 'Age' affecting model prediction

Let's take a look at how the `Age` feature affects model predictions.

PDP for the Age feature

Just like we observed before. the middle-aged people have a slightly higher shap value, pushing the model's prediction decisions to say that these individuals make more money as compared to younger or older people

PDP of 'Education-Num' affecting model prediction

Let's take a look at how the `Education-Num` feature affects model predictions.

PDP for the Education-Num feature

Higher education levels have higher shap values, pushing the model's prediction decisions to say that these individuals make more money as compared to people with lower education levels.

PDP for the Relationship feature

Just like we observed during the model prediction explanations, married people (husband or wife) have a slightly higher shap value, pushing the model's prediction decisions to say that these individuals make more money as compared to other folks!

PDP of 'Capital Gain' affecting model prediction

Let's take a look at how the `Capital Gain` feature affects model predictions.

PDP for the Capital Gain feature

Two-way PDP showing interactions between features 'Age' and 'Capital Gain' and their effect on making more than $50K

The vertical dispersion of SHAP values at a single feature value is driven by interaction effects, and another feature is chosen for coloring to highlight possible interactions. Here we are trying to see interactions between `Age` and `Capital Gain` and also their effect on the SHAP values which lead to the model predicting if the person will make more money or not, with the help of a two-way partial dependence plot.

Two-way PDP showing effects of the Age and Capital Gain features

Interesting to see higher the higher capital gain and the middle-aged folks (30–50) having the highest chance of making more money!

Two-way PDP showing interactions between features 'Education-Num' and 'Relationship' and their effect on making more than $50K

Here we are trying to see interactions between **`Education-Num`** and **`Relationship`** and also their effect on the SHAP values which lead to the model predicting if the person will make more money or not, with the help of a two-way partial dependence plot.

Two-way PDP showing effects of the Education-Num and Relationship features

This is interesting because both the features are similar in some context, we can see typically married people with relationship status of either husband or wife having the highest chance of making more money!

Two-way PDP showing interactions between features 'Marital Status' and 'Relationship' and their effect on making more than $50K

Here we are trying to see interactions between `Marital Status` and `Relationship` and also their effect on the SHAP values which lead to the model predicting if the person will make more money or not, with the help of a two-way partial dependence plot.

Two-way PDP showing effects of the Marital Status and Relationship features

Interesting to see higher the higher education level and the husband or wife (married) folks having the highest chance of making more money!

Two-way PDP showing interactions between features 'Age' and 'Hours per week' and their effect on making more than $50K

Here we are trying to see interactions between `Age` and `Hours per week` and also their effect on the SHAP values which lead to the model predicting if the person will make more money or not, with the help of a two-way partial dependence plot.

Two-way PDP showing effects of the Age and Hours per week features

Nothing extra-ordinary here, middle-aged people working the most make the most money!

Conclusion

If you are reading this, I would like to really commend your efforts on going through this huge and comprehensive tutorial on machine learning model interpretation. This

concepts and techniques we learnt in *Part 2*, in this article, we actually implemented them all on a complex machine learning ensemble model trained on a real-world dataset. I encourage you to try out some of these frameworks with your own models and datasets and explore the world of model interpretation!

What's next?

In Part 4 of this series, we will be looking at a comprehensive guide to building interpreting models on unstructured data like text and maybe even deep learning models!

- Hands-on Model Interpretation on Unstructured Datasets
- Advanced Model Interpretation on Deep Learning Models

Stay tuned for some interesting content!

> ***Note:*** *There are a lot of rapid developments in this area including a lot of new tools and frameworks being released over time. In case you want me to cover any other popular frameworks, feel free to reach out to me. I'm definitely interested and will be starting by taking a look into H2O's model interpretation capabilities some time in the future.*

The code used in this article is available on **my GitHub** and also as an interactive ***Jupyter Notebook***.

Check out ***'Part 1 — The Importance of Human Interpretable Machine Learning'*** which covers the what and why of human interpretable machine learning and the need and importance of model interpretation along with its scope and criteria in case you haven't!

Also ***Part 2 — Model Interpretation Strategies'*** which covers the how of human interpretable machine learning where we look at essential concepts pertaining to

Have feedback for me? Or interested in working with me on research, data science, artificial intelligence or even publishing an article on ***TDS***? You can reach out to me on **LinkedIn**.