```
!pip install shap pyDOE2
from IPython.core.display import display, HTML
import regex as re
import lightgbm
import pandas as pd
import shap
import sklearn

import xgboost as xgb
from sklearn.model_selection import train_test_split
import lightgbm as lgb
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: pyDOE2 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dis
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/py
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dis
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pack
```

```
!pip install lime
```

```
Requirement already satisfied: lime in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.11
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.11
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/pythor
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.1
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/dist
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.11/c
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/c
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/c
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pacl
```

Patch to match style consistency

```
"""
Explanation class, with visualization functions.
"""
import lime
from io import open
import os
import os.path
import json
import string
import numpy as np

from lime.exceptions import LimeError

from sklearn.utils import check_random_state

def patch_show_in_notebook(self,
                      labels=None,
                      predict_proba=True,
                      show_predicted_value=True,
                      **kwargs):
    """Shows html explanation in ipython notebook.

    See as_html() for parameters.
    This will throw an error if you don't have IPython installed"""

    from IPython.core.display import display, HTML
    the_html = self.as_html(labels=labels,
                          predict_proba=predict_proba,
                          show_predicted_value=show_predicted_value,
                          **kwargs)

    the_html = the_html.replace("Prediction probabilities","\t\t\t\t        predi

    display(HTML(the_html))

lime.explanation.Explanation.show_in_notebook = patch_show_in_notebook
```

```
from lime.lime_tabular import LimeTabularExplainer
import collections
import copy
from functools import partial
import json
import warnings
```

```
import numpy as np
import scipy as sp
import sklearn
import sklearn.preprocessing
from sklearn.utils import check_random_state
from pyDOE2 import lhs
from scipy.stats.distributions import norm

from lime.discretize import QuartileDiscretizer
from lime.discretize import DecileDiscretizer
from lime.discretize import EntropyDiscretizer
from lime.discretize import BaseDiscretizer
from lime.discretize import StatsDiscretizer
from lime.lime_tabular import TableDomainMapper

from lime.lime_tabular import explanation
from lime.lime_tabular import lime_base

def patch_explain_instance(self,
                      data_row,
                      existing_values,
                      predict_fn,
                      labels=(1,),
                      top_labels=None,
                      num_features=10,
                      num_samples=5000,
                      distance_metric='euclidean',
                      model_regressor=None,
                      sampling_method='gaussian'):
    """Generates explanations for a prediction.

    First, we generate neighborhood data by randomly perturbing features
    from the instance (see __data_inverse). We then learn locally weighted
    linear models on this neighborhood data to explain each of the classes
    in an interpretable way (see lime_base.py).

    Args:
        data_row: 1d numpy array or scipy.sparse matrix, corresponding to a row
        predict_fn: prediction function. For classifiers, this should be a
            function that takes a numpy array and outputs prediction
            probabilities. For regressors, this takes a numpy array and
            returns the predictions. For ScikitClassifiers, this is
            `classifier.predict_proba()`. For ScikitRegressors, this
            is `regressor.predict()`. The prediction function needs to work
```

```
            on multiple feature vectors (the vectors randomly perturbed
            from the data_row).
        labels: iterable with labels to be explained.
        top_labels: if not None, ignore labels and produce explanations for
            the K labels with highest prediction probabilities, where K is
            this parameter.
        num_features: maximum number of features present in explanation
        num_samples: size of the neighborhood to learn the linear model
        distance_metric: the distance metric to use for weights.
        model_regressor: sklearn regressor to use in explanation. Defaults
            to Ridge regression in LimeBase. Must have model_regressor.coef_
            and 'sample_weight' as a parameter to model_regressor.fit()
        sampling_method: Method to sample synthetic data. Defaults to Gaussian
            sampling. Can also use Latin Hypercube Sampling.

    Returns:
        An Explanation object (see explanation.py) with the corresponding
        explanations.
    """
    if sp.sparse.issparse(data_row) and not sp.sparse.isspmatrix_csr(data_row):
        # Preventative code: if sparse, convert to csr format if not in csr forma
        data_row = data_row.tocsr()
    data, inverse = self._LimeTabularExplainer__data_inverse(data_row, num_sample
    if sp.sparse.issparse(data):
        # Note in sparse case we don't subtract mean since data would become dens
        scaled_data = data.multiply(self.scaler.scale_)
        # Multiplying with csr matrix can return a coo sparse matrix
        if not sp.sparse.isspmatrix_csr(scaled_data):
            scaled_data = scaled_data.tocsr()
    else:
        scaled_data = (data - self.scaler.mean_) / self.scaler.scale_
    distances = sklearn.metrics.pairwise_distances(
            scaled_data,
            scaled_data[0].reshape(1, -1),
            metric=distance_metric
    ).ravel()

    yss = predict_fn(inverse)

    # for classification, the model needs to provide a list of tuples - classes
    # along with prediction probabilities
    if self.mode == "classification":
        if len(yss.shape) == 1:
            raise NotImplementedError("LIME does not currently support "
                                      "classifier models without probability "
```

```
                                        "scores. If this conflicts with your "
                                        "use case, please let us know: "
                                        "https://github.com/datascienceinc/lime/iss
        elif len(yss.shape) == 2:
            if self.class_names is None:
                self.class_names = [str(x) for x in range(yss[0].shape[0])]
            else:
                self.class_names = list(self.class_names)
            if not np.allclose(yss.sum(axis=1), 1.0):
                warnings.warn("""
                Prediction probabilties do not sum to 1, and
                thus does not constitute a probability space.
                Check that you classifier outputs probabilities
                (Not log probabilities, or actual class predictions).
                """)
        else:
            raise ValueError("Your model outputs "
                                "arrays with {} dimensions".format(len(yss.shape)))

    # for regression, the output should be a one-dimensional array of predictions
    else:
        try:
            if len(yss.shape) != 1 and len(yss[0].shape) == 1:
                yss = np.array([v[0] for v in yss])
            assert isinstance(yss, np.ndarray) and len(yss.shape) == 1
        except AssertionError:
            raise ValueError("Your model needs to output single-dimensional \
                numpyarrays, not arrays of {} dimensions".format(yss.shape))

        predicted_value = yss[0]
        min_y = min(yss)
        max_y = max(yss)

        # add a dimension to be compatible with downstream machinery
        yss = yss[:, np.newaxis]

    feature_names = copy.deepcopy(self.feature_names)
    if feature_names is None:
        feature_names = [str(x) for x in range(data_row.shape[0])]

    if sp.sparse.issparse(data_row):
        values = self.convert_and_round(data_row.data)
        feature_indexes = data_row.indices
    else:
        values = self.convert_and_round(data_row)
```

```
            feature_indexes = None

        for i in self.categorical_features:
            if self.discretizer is not None and i in self.discretizer.lambdas:
                continue
            name = int(data_row[i])
            if i in self.categorical_names:
                name = self.categorical_names[i][name]
            feature_names[i] = '%s\t\t\t' % (feature_names[i])
            values[i] = name
        categorical_features = self.categorical_features

        discretized_feature_names = None
        if self.discretizer is not None:
            categorical_features = range(data.shape[1])
            discretized_instance = self.discretizer.discretize(data_row)
            discretized_feature_names = copy.deepcopy(feature_names)
            for f in self.discretizer.names:
                discretized_feature_names[f] = self.discretizer.names[f][int(
                        discretized_instance[f])]

        domain_mapper = TableDomainMapper(feature_names,
                                          values,
                                          scaled_data[0],
                                          categorical_features=categorical_features,
                                          discretized_feature_names=discretized_featu
                                          feature_indexes=feature_indexes)
        ret_exp = explanation.Explanation(domain_mapper,
                                          mode=self.mode,
                                          class_names=self.class_names)

        if self.mode == "classification":
            ret_exp.predict_proba = yss[0]
            if top_labels:
                labels = np.argsort(yss[0])[-top_labels:]
                ret_exp.top_labels = list(labels)
                ret_exp.top_labels.reverse()
        else:
            ret_exp.predicted_value = predicted_value
            ret_exp.min_value = min_y - 10
            ret_exp.max_value = max_y + 10
            labels = [0]
        for label in labels:
            ret_exp.score = {}
            ret_exp.local_pred = {}
```

```
        (ret_exp.intercept[label],
         ret_exp.local_exp[label],
         ret_exp.score[label],
         ret_exp.local_pred[label]) = self.base.explain_instance_with_data(
                scaled_data,
                yss,
                distances,
                label,
                num_features,
                model_regressor=model_regressor,
                feature_selection=self.feature_selection)
    list_to_sort = existing_values
    sorted_list = sorted([abs(float(val)) for val in list_to_sort])
    sorted_list.reverse()
    final_list = []
    final_vals = []

    i_list = [j for j in range(len(values))]
    for k in range(len(existing_values)):
      for l in range(len(existing_values)):
        if (abs(float(list_to_sort[l])) == sorted_list[k]):
          final_list.append(list_to_sort[l])
          final_vals.append(i_list[l])
    for i in range(len(existing_values)):
      ret_exp.local_exp[1][i] = (final_vals[i], final_list[i])

    if self.mode == "regression":
        ret_exp.intercept[1] = ret_exp.intercept[0]
        ret_exp.local_exp[1] = [x for x in ret_exp.local_exp[0]]
        ret_exp.local_exp[0] = [(i, -1 * j) for i, j in ret_exp.local_exp[1]]
    return ret_exp
LimeTabularExplainer.explain_instance = patch_explain_instance
```

Set up tutorial examples

Start by training the "should you bring an umbrella?" model

```
preX = pd.read_csv("Umbrella.csv")
preX = preX.sample(frac=1)
X_display = preX.iloc[:,:-1]
y_display = preX.iloc[:,-1]

PRECIPITATION = {
```

```
    "none": 0,
    "drizzle": 1,
    "rain": 2,
    "snow": 3,
    "sleet": 4,
    "hail": 5
}

y = y_display
X = X_display
X = X.replace({"Precipitation":PRECIPITATION})

X_train = X.iloc[:300]
y_train = y.iloc[:300]

X_test = X.iloc[300:]
y_test = y.iloc[300:]

d_train = lightgbm.Dataset(X_train, label=y_train)
d_test = lightgbm.Dataset(X_test, label=y_test)

params = {
    "max_bin": 512,
    "learning_rate": 0.05,
    "boosting_type": "gbdt",
    "objective": "binary",
    "metric": "binary_logloss",
    "num_leaves": 10,
    "verbose": -1,
    "min_data": 100,
    "boost_from_average": True,
    "keep_training_booster": True
}

#model = lgb.train(params, d_train, 10000, valid_sets=[d_test]) #early_stopping_r
model = lightgbm.LGBMClassifier(max_bin= 512,
    learning_rate= 0.05,
    boosting_type= "gbdt",
    objective= "binary",
    metric= "binary_logloss",
    num_leaves= 10,
    verbose= -1,
    min_data= 100,
    boost_from_average= True)
model.fit(X_train, y_train)
```

```
<ipython-input-56-59731d8556ad>:17: FutureWarning: Downcasting behavior in `re
  X = X.replace({"Precipitation":PRECIPITATION})
```

```
LGBMClassifier

LGBMClassifier(boost_from_average=True, learning_rate=0.05, max_bin=512,
               metric='binary_logloss', min_data=100, num_leaves=10,
               objective='binary', verbose=-1)
```

Find the location of one of the two tutorial examples

```
print(X.loc[(X['Precipitation'] == 5) & (X['Temperature'] == 23) & (X['Wind(mph)']
print(X.loc[(X['Precipitation'] == 0) & (X['Temperature'] == 70) & (X['Wind(mph)']
theloc = X.index.get_loc(330)
```

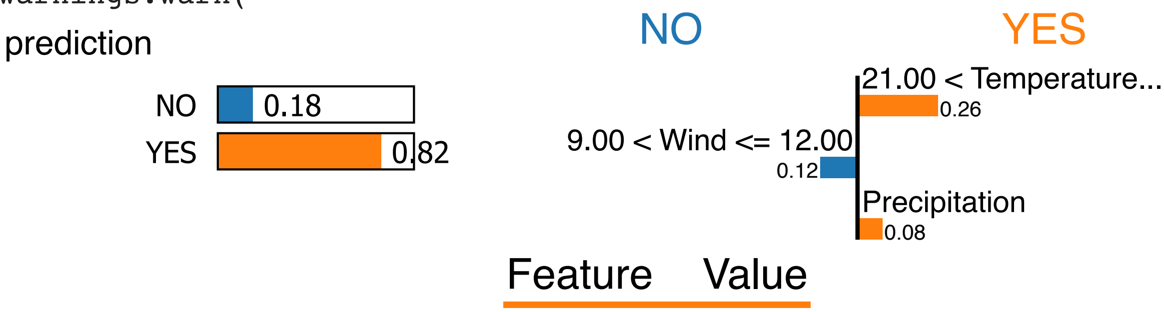
```
     Precipitation  Temperature  Wind(mph)
330              5           23         10
    Precipitation  Temperature  Wind(mph)
96              0           70         30
```

Generate a tutorial explanation

```
import lime
from lime import lime_tabular
from lime.lime_tabular import LimeTabularExplainer
limexplainer = LimeTabularExplainer(X.to_numpy(), training_labels=y, mode='classi
            feature_names=["Precipitation", "Temperature", "Wind"],
                categorical_features=[0],
            categorical_names={0:["none", "drizzle", "sleet", "snow", "sleet", "h
            class_names=["NO","YES"],  discretizer='decile', kernel_width=0.85, r
```

```
exp = limexplainer.explain_instance(X.iloc[theloc].to_numpy(), np.array([0.08, 0.
exp.save_to_file("./saved_fig_intro_1")
exp.show_in_notebook()#show_table=True, show_all=True)
```

```
Intercept 0.605056199451296
Prediction_local [0.76466031]
Right: 0.8230855215029175
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: Futu
  warnings.warn(
```



Loan Instances

Edit and prepare dataset

```
# load dataset
X,y = shap.datasets.adult()
X_display,y_display = shap.datasets.adult(display=True)

EDUCATION_NUM = {
    16.0: "Doctorate",
    15.0: "Prof. School",
    14.0: "Masters",
    13.0: "Bachelors",
    12.0: "Some College",
    11.0: "Associate", #Assoc-acdm
    10.0: "Vocational", #Assoc-voc
    9.0: "HS grad",
    8.0: "12th",
    7.0: "11th",
    6.0: "10th",
    5.0: "9th",
    4.0: "7th-8th",
    3.0: "5th-6th",
    2.0: "1st-4th",
```

```
    1.0: "Preschool"
}

OCCUPATION_NUM = {
    "Tech-support": "Tech Support",
    "Craft-repair": "Craft/Repair",
    "Other-service": "Other Service",
    "Sales": "Sales",
    "Exec-managerial": "Exec. Managerial",
    "Prof-specialty": "Prof. Specialty",
    "Handlers-cleaners": "Handler/Cleaner",
    "Machine-op-inspct": "Machine Op. Inspector",
    "Adm-clerical": "Admin. Clerical",
    "Farming-fishing": "Farming/Fishing",
    "Transport-moving": "Transport/Moving",
    "Priv-house-serv": "Private House Service",
    "Protective-serv": "Protective Service",
    "Armed-Forces": "Armed Forces"

}
X_display = X_display.replace({"Education-Num":EDUCATION_NUM})
X_display = X_display.replace({"Occupation":OCCUPATION_NUM})
X = X.rename(columns={"Education-Num": "Education"})
X_display = X_display.rename(columns={"Education-Num": "Education"})#, "Hours per

X = X.drop(['Capital Loss', 'Capital Gain', 'Race', 'Relationship', 'Country', 'W
X_display = X_display.drop(['Capital Loss', 'Capital Gain', 'Race', 'Relationship

# create a train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
d_train = lgb.Dataset(X_train, label=y_train)
d_test = lgb.Dataset(X_test, label=y_test)
```

Train the model

```
params = {
    "max_bin": 512,
    "learning_rate": 0.05,
    "boosting_type": "gbdt",
    "objective": "binary",
    "metric": "binary_logloss",
    "num_leaves": 10,
    "verbose": -1,
    "min_data": 100,
    'objective':'multi:softprob',
    "boost_from_average": True
}

params_xgb={
    'base_score':0.5,
    'learning_rate':0.05,
    'max_depth':5,
    'min_child_weight':100,
    'n_estimators':200,
    'num_class': 2,
    'nthread':-1,
    'objective':'multi:softprob',
    'seed':2018,
    'eval_metric':'auc'
}

model = lgb.LGBMClassifier(max_bin= 512,
    learning_rate= 0.05,
    boosting_type= "gbdt",
    objective= "binary",
    metric= "binary_logloss",
    num_leaves= 10,
    verbose= -1,
    min_data= 100,
    boost_from_average= True)
model.fit(X_train, y_train)
```

```
LGBMClassifier

LGBMClassifier(boost_from_average=True, learning_rate=0.05, max_bin=512,
               metric='binary_logloss', min_data=100, num_leaves=10,
               objective='binary', verbose=-1)
```

Our 7 loan application instances

```
#val = 610 # Woman Side-by-side
#val = 11116 # Man Side-by-side
#val = 32353 # Man 3
#val = 217 # Man 2
#val = 15040 # Man 1
#val = 32429 # Woman 3
val = 32556 # Woman 2
#val = 91#91 # Woman 1

theloc = val
```

Generate LIME Explanation

```
import lime
from lime import lime_tabular
from lime.lime_tabular import LimeTabularExplainer

limexplainer = LimeTabularExplainer(X.to_numpy(), training_labels=y, mode='classi
            feature_names=[ "Age","Education","Occupation", "Sex", "Hours worked 
            categorical_features=[1,2,3],
            categorical_names={1:["None","Preschool", "1st-4th", "5th-6th", "7th-
            class_names=["NO","YES"],  discretizer='decile', kernel_width=0.85, r
```

```
#shap_values_standin0 = pd.Series({'Age': 0.0307, 'Education': -0.0287, 'Occupati
shap_values_standin0 = pd.Series({'Age': -0.14, 'Education': 0.0416, 'Occupation'
#shap_values_standin0 = pd.Series({'Age': 0.1209, 'Education': 0.3008, 'Occupatio
#shap_values_standin0 = pd.Series({'Age': -0.2119, 'Education': 0.0011, 'Occupati
#shap_values_standin0 = pd.Series({'Age': 0.0565, 'Education': 0.1427, 'Occupatio
#shap_values_standin0 = pd.Series({'Age': -0.0012, 'Education': -0.189, 'Occupati
#shap_values_standin0 = pd.Series({'Age': 0.0774, 'Education': 0.1962, 'Occupatio
#shap_values_standin0 = pd.Series({'Age': 0.0668, 'Education': 0.1619, 'Occupatio

#exp = limexplainer.explain_instance(X.iloc[theloc], [shap_values_standin0['Age']

exp = limexplainer.explain_instance(X.iloc[theloc], shap_values_standin0, model.p
exp.show_in_notebook()#show_table=True, show_all=True)
```

```
/usr/local/lib/python3.11/dist-packages/lime/discretize.py:110: FutureWarning:
  ret[feature] = int(self.lambdas[feature](ret[feature]))
/usr/local/lib/python3.11/dist-packages/lime/discretize.py:110: FutureWarning:
  ret[feature] = int(self.lambdas[feature](ret[feature]))
/usr/local/lib/python3.11/dist-packages/lime/lime_tabular.py:544: FutureWarnir
  binary_column = (inverse_column == first_row[column]).astype(int)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: Futu
  warnings.warn(
<ipython-input-55-aaf2d5e87ad8>:147: FutureWarning: Series.__getitem__ treatir
  name = int(data_row[i])
/usr/local/lib/python3.11/dist-packages/lime/discretize.py:110: FutureWarning:
  ret[feature] = int(self.lambdas[feature](ret[feature]))
/usr/local/lib/python3.11/dist-packages/lime/discretize.py:110: FutureWarning:
  ret[feature] = int(self.lambdas[feature](ret[feature]))
<ipython-input-55-aaf2d5e87ad8>:161: FutureWarning: Series.__getitem__ treatir
  discretized_instance[f])]
<ipython-input-55-aaf2d5e87ad8>:207: FutureWarning: Series.__getitem__ treatir
  if (abs(float(list_to_sort[l])) == sorted_list[k]):
<ipython-input-55-aaf2d5e87ad8>:208: FutureWarning: Series.__getitem__ treatir
  final_list.append(list_to_sort[l])
Intercept 0.24502110197409716
Prediction_local [0.10012927]
Right: 0.07751175939332404
```