IMPLEMENTACIÓN DE GRÁFICOS SHAP

Visión general de la solución planteada

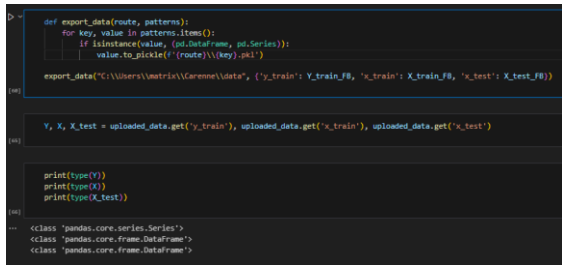
FLUJO INICIAL: MANEJO DE DATOS

La función recibe un diccionario con la clave y el respectivo objeto a exportar. Es crucial seleccionar una clave que lo identifique inequívocamente, ya que se relacionará al momento de invocar la función "load_data".

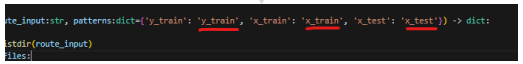
Exportar conjunto de datos: Opcional

1

Para facilitar la gestión del control de datos, es esencial exportarlos a un formato que permita su manipulación sin implicar todo el preprocesamiento cada vez que desee utilizar la función.



Se utilizó el “to_pickle” con el fin de mantener el formato original de los datos, cosa que no era posible con “to_csv”.

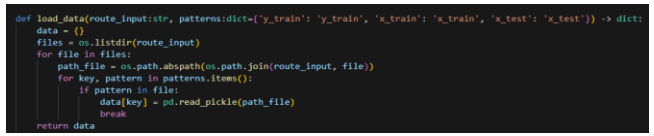


En el parámetro patterns se debe especificar la clave con la que se guardo el dataset localmente.

Carga de datos

2

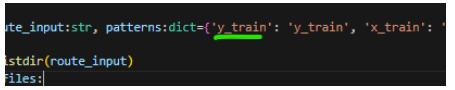
La función está diseñada para funcionar de manera independiente a la exportación, por lo que buscará en la ruta proporcionada a su llamada los conjuntos de datos requeridos en el procedimiento. Devolverá un diccionario con clave: objeto para ser extraídos con la función "get".



Obtener valores

3

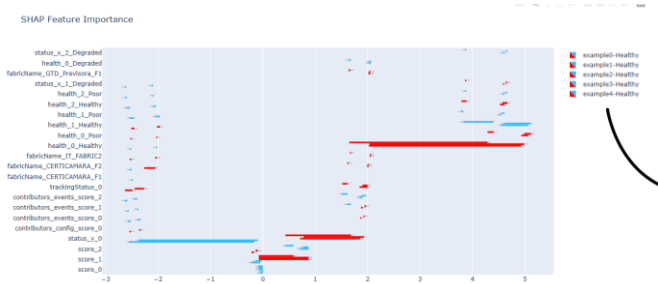
Mediante el uso de get y las claves establecidas en el diccionario retornado por “load_data” es posible asignar los datos a las variables correspondientes.



/manual/data

Almacenará los dataset exportados.

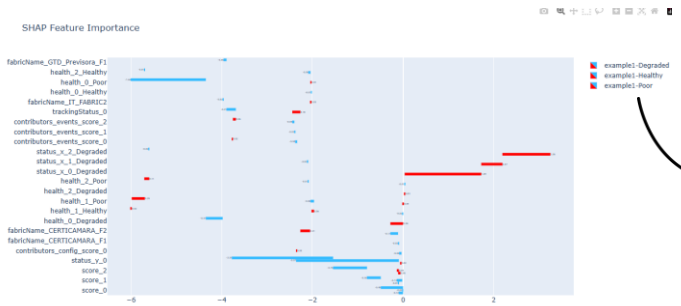
TIPOS DE VISUALIZACIÓN : VALORES SHAP



Mod 1

Los valores SHAP se generarán por la clase predicha de cada muestra.

Permite seleccionar la forma en que se mostrarán los valores SHAP de cada muestra.



Mod 2

Los valores SHAP serán generados para todas las clases de cada muestra.

En la llamada de la función principal, deberá pasar a la función "select_examples" las muestras que desea graficar. Existen las opciones: límite, todos y ejemplos específicos

```python
def select_examples(mode: str, x_test, range_example:int=5, specific_example:list=[0, 1]) -> list:
    if mode == "limit":
        id_example = [ide for ide in range(range_example)]
    elif mode == "total":
        id_example = [ide for ide in range(len(x_test))]
    elif mode == "specific":
        id_example = [ide for ide in specific_example]
    return id_example
```

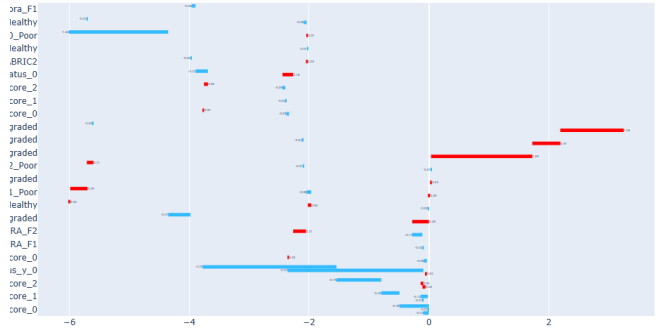
En la creación del objeto de la clase, el parámetro “viz_type” es el que se encarga de manejar los modos; el valor True dejará activado por defecto el mod 1.

```python
class Graph_Shap:
    def __init__(self, x_train:pd.DataFrame, y_train:pd.Series, x_test:pd.DataFrame, viz_type:bool=True, route_html:str="",
    viz_graph:bool=False):
        self.x_train = x_train
        self.x_test = x_test
        self.y_train = y_train
        self.viz_type_shap = viz_type # Controla el tipo de visualización de los values shap de cada muestra, ya sea por clase
        predicha o por cada tipo de clase
        self.viz_graph = viz_graph # Controla el formato de salida del gráfico, de manera individual o simplificada en un sola figura
        self.unique_classes = y_train.unique().tolist()
        self.route_html = route_html
        self.fig = None
```

Cuando se establece en False, el mod 2 será el activado.

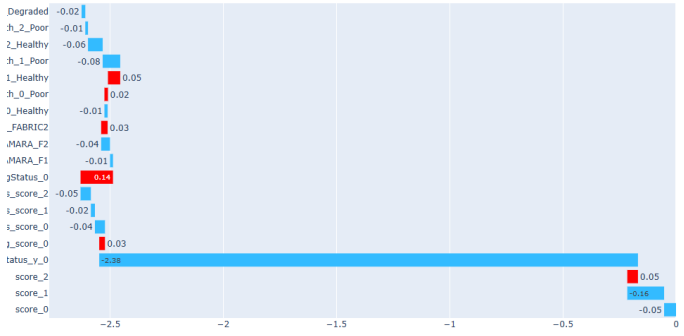
```python
if self.viz_type_shap:
    class_id = self.shap_value_id.base_values.argmax() # Conseguir la clase predicha
    shap_df = self.__importance(self.shap_value_id, class_id) # Dejar solo las caracteristicas importantes que
    benefician a la clase predicha
    self.fig = self.__settings_bars(type_shap, shap_df, f'example{example}-{self.unique_classes[class_id]}')
else:
    for index, class_name in enumerate(self.unique_classes): # Valores shap para cada clase del conjunto en cada
    muestra
        shap_df = self.__importance(self.shap_value_id, index)
        self.fig = self.__settings_bars(type_shap,shap_df, f'example{example}-{class_name}')
```

TIPOS DE VISUALIZACIÓN : GRÁFICOS SHAP



Viz 1

Los valores SHAP serán utilizados en una sola figura.



Viz 2

En este modo, cada valor SHAP tendrá su propia figura.

Permite establecer la descarga de los gráficos SHAP en un solo HTML global e individual.

Por defecto está establecida en True dejando activado el modo Viz 1.

```python
class Graph_Shap:
    def __init__(self, x_train:pd.DataFrame, y_train:pd.Series, x_test:pd.DataFrame, viz_type:bool=True, route_html:str="", viz_graph:bool=True):
        self.x_train = x_train
        self.x_test = x_test
        self.y_train = y_train
        self.viz_type_shap = viz_type # Controla el tipo de visualización de los values shap de cada muestra, ya sea por clase predicha o por cada
        self.viz_graph = viz_graph # Controla el formato de salida del gráfico, de manera individual o simplificada en un sola figura
        self.unique_classes = y_train.unique().tolist()
        self.route_html = route_html
        self.fig = None
```

El método “manage_figure” gestiona el control de los cambios en la figura “go.figure”.

```python
def __manage_figure(self):
    if self.viz_graph:
        if not hasattr(self, 'fig') or self.fig is None:
            self.fig = go.Figure()
    else:
        self.fig = go.figure()
    return self.fig

def graphics_shap(self, shap_values: shap._explanation.Explanation, color_scheme: str, id_example: [list, None] = [0],
type_shap:str = "waterfall") -> go.Figure: # type: ignore
    self.color_scheme = color_scheme
    try:
        charFilter = id_example if id_example is not None else range(len(shap_values))
        for example in charFilter:
            self.fig = self.__manage_figure()
            self.shap_value_id = shap_values[example]
```

Los gráficos pueden exportarse con formato de SHAP y con el formato con el que se desea manejar la figura.

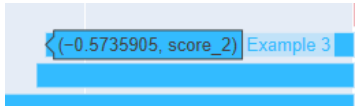
```python
            self.fig = self.__settings_bars(type_shap, shap_df, f'example{example}-{self.unique_classes[class_id]}')
    else:
        for index, class_name in enumerate(self.unique_classes): # Valores shap para cada clase del conjunto en cada muestra
            shap_df = self.__importance(self.shap_value_id, index)
            self.fig = self.__settings_bars(type_shap,shap_df, f'example{example}-{class_name}')

    if not self.viz_graph:
        self.export_graph(f'{str(example)}-mod2-viz1') if self.viz_type_shap else self.export_graph(f'{str(example)}-mod2-viz2')

if self.viz_graph:
    self.export_graph(f'{str(charFilter)}-mod1-viz1') if self.viz_type_shap else self.export_graph(f'{str(charFilter)}-mod1-viz2')
```

TIPOS DE VISUALIZACIÓN : CONFIGURACIÓN DEL GRÁFICO

```python
def __settings_bars(self, type_shap, shap_df: pd.DataFrame, example_name: str) -> go.Figure: # type: ignore
    increasing, decreasing = self.__get_scheme()
    settings = {
        "waterfall": {
            "name": example_name,
            "orientation": 'h',
            "measure": ['relative'] * len(shap_df),
            "x": shap_df['value'],
            "y": shap_df['feature'],
            "text": shap_df['value'].apply(lambda x: f'{x:.2f}'),
            "connector": {"line": {"width": 0}},
            "increasing": dict(marker=dict(color=increasing, line=dict(width=0))),
            "decreasing": dict(marker=dict(color=decreasing, line=dict(width=0)))
        },
        "bar": {
            "title": example_name,
            "orientation": 'h',
            "x": shap_df['value'],
            "y": shap_df['feature'],
            "text": shap_df['value'].apply(lambda x: f'{x:.2f}'),
            "marker": {
                "color": shap_df['value'].apply(lambda x: increasing if x > 0 else decreasing),
                "line": dict(width=0)
                }
            }
    }
    if type_shap == "waterfall":
        self.fig.add_trace(go.Waterfall(**settings['waterfall']))
    elif type_shap == "bar":
        self.fig.add_trace(go.Bar(**settings['bar']))
    else:
        raise ValueError(f"Type of graph not supported: {type_shap}")
    return self.fig
```

Método diseñado con el objetivo de ampliar los tipos de gráficos SHAP, en la que se proporciona una configuración predeterminada. En caso de agregar otro tipo de gráfico, debe realizarse en el recuadro rojo. Esta configuración es para los trazos añadidos al gráfico. Existen más configuraciones base en la función __settings_shap para el gráfico resultante.

Visualización en waterfall.



Visualización en barras.

```python
def __get_scheme(self) -> tuple: # type: ignore
    try:
        if isinstance(self.color_scheme, str):
            colors = getattr(px.colors.diverging, self.color_scheme)
            return colors[0], colors[1]
        elif isinstance(self.color_scheme, list) and len(self.color_scheme) == 2:
            return self.color_scheme[0], self.color_scheme[1]
    except ValueError:
        print("The color scheme must have hexadecimal values or one of these supported schemes: 'RdBu', 'GnPR', 'CyPU', 'PkYg', 'DrDb', 'LpLb', 'Y1Dp', 'OrId'")
```

Los colores del grafico son manejados por el método “get_scheme” el cual puede recibir dos tipos de valores:

- Lista con los colores específicos.
- Nombre de un esquema valido.

```python
if __name__ == '__main__':
    viz_type = True
    current_path = os.getcwd()
    route_html = os.path.abspath(os.path.join(current_path)) + "\\graphis"

    uploaded_data = load_data(f'{current_path}\\data')
    Y, X, X_test = uploaded_data.get('y_train'), uploaded_data.get('x_train'), uploade

    color_scheme = ['rgb(255, 0, 0)', '#3B8BFF']
```

La lista puede manejar formato rgb() o hexadecimal.

El parámetro connector en la configuración predeterminada de los trazos, si no se establece su tamaño en 0, el gráfico resultante tendría esta representación.

RECOMENDACIONES FALTANTES

```python
def __mapping_labels(self, Y_train) -> pd.Series:
    if Y_train.dtype == "object":
        map = {class_name: idx for idx, class_name in enumerate(self.unique_classes)}
        return Y_train.replace(map) # Downcasting behavior in `replace` is deprecated and will be removed in a future version.
    return Y_train

def __training(self, classifier, X:pd.DataFrame, Y:pd.DataFrame, **kwargs:dict) -> xgboost.XGBClassifier:
    try:
        Y_map = self.__mapping_labels(Y)
        return classifier(**kwargs).fit(X, Y_map)
    except Exception as e:
        print(f'Error in training the model: {e}')
```

Es crucial que el modelo xgboost.XGBClassifier se entrene con un conjunto de datos donde las etiquetas tengan un formato numérico. Si este no es el caso, la función "mappin_labels" reemplazará el formato con identificadores numéricos.

```
---> 98     model.fit(X_train_FB, Y_train_FB)
       model = XGBClassifier(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
       X_train_FB =       score_0  score_1  score_2  status_y_0  status_y_1  status_y_2  \
...
   1498 if callable(self.objective):

ValueError: Invalid classes inferred from unique values of `y`.  Expected: [0 1 2], got ['Degraded' 'Healthy' 'Poor']

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```python
def __importance(self, shap_values:shap.Explainer, class_index:int, threshold:float = 0.01) -> pd.DataFrame:
    values_class = shap_values.values[:, class_index]
    shap_df = pd.DataFrame({
        'feature': shap_values.feature_names,
        'value': values_class
    })
    shap_df = shap_df[shap_df['value'].abs() > threshold]
    return shap_df
```

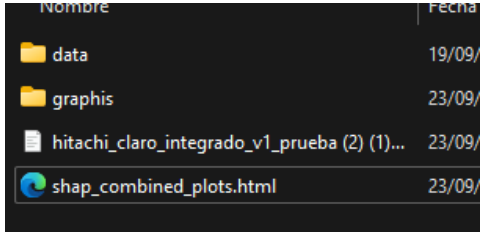
En caso de que no se deseen filtrar las características más importantes, simplemente se debe modificar la función "importance“.

MANEJO DEL HTML RESULTANTE

```python
class Html_master:
    def __init__(self, html_parts) -> None:
        self.html_parts = html_parts
        self.html_master = """
            <!DOCTYPE html>
            <html lang="en">
            <head>
                <meta charset="UTF-8">
                <meta name="viewport" content="width=device-width, initial-scale=1.0">
                <title>SHAP Plots</title>
                <style>{style}</style>
                <script>
                    function showGraph(graphId) {
                        var graphs = document.getElementsByClassName('shap-graph');
                        for (var i = 0; i < graphs.length; i++) {
                            graphs[i].style.display = 'none';  // Ocultar todos
                        }
                        document.getElementById(graphId).style.display = 'block';  // Mostrar seleccionado
                    }
                </script>
            </head>
            <body>
                <h1>Selecciona un gráfico SHAP</h1>
                <select id="graph-selector" onchange="showGraph(this.value)">
                    {options}
                </select>

                {graphs}
            </body>
            </html>
            """
```

Esta clase permite la creación de un HTML maestro que concatenará los gráficos SHAP exportados previamente con el uso de la clase Graph_Shap. Mediante una barra desplegable, el usuario puede cambiar de gráfico.

Estructura final de directorios




Matrix
CPM SOLUTIONS