diabetes-lime-shap

May 13, 2021

1 Explicabilidad de modelos de aprendizaje con LIME y SHAP

1.1 Introducción

La diabetes es una enfermedad que ocurre cuando la concentración de glucosa en sangre supera un umbral.

Los seres humanos tenemos una hormona llamada insulina que permite transportar esta glucosa a las células, que la usarán para obtener energía finalmente y poder realizar sus funciones.

Si la glucosa permanece en la sangre y no es transportada al interior de las células, se dice que esa persona es diabética y puede concluir en la amputación de zonas corporales o directamente la muerte.

Nosotros proponemos un ejercicios práctico para aplicar las dos técnicas de explicabilidad que hemos visto, es decir, LIME y SHAP, sobre modelos de aprendizaje automático que han aprendido sobre de datos extraido del repositorio de aprendizaje automático de la Universidad de California, Irvine (UCI) que incluyen 8 características como la edad, el grosor de la piel o el índice de masa corporal soble un fragmento de población de un grupo indígena que vive en el estado de Arizona, Estados Unidos.

1.2 Características

Este conjunto de datos consta de varios parámetros médicos y un parámetro dependiente que es la clase, es decir, si es diabético o no. Este conjunto de datos está formado por una población íntegramente femenina luego el sexo no es una dimensión del conjunto de datos.

Encontramos en total 9 dimensiones donde 8 son parámetros independientes, con un total de 768 observaciones, donde 268 padecen diabetes y 500 no.

- El primer parámetro que se estudia es el número de veces que han estado embarazadas. El segundo parámetro que se estudia es el resultado de una prueba de tolerancia a la glucosa por vía oral. Esta prueba indica cómo el cuerpo es capaz de transportar esa glucosa al interior de los diferentes tipos celulares, y es uno de los test más comunes para diagnosticar la diabetes a nivel mundial. Este examen consiste en tomar una muestra de sangre habiendo realizado ayuno. Una persona sin diabetes debería de tener una concentración menor a los 110 mg/dL, y una muestra que supere los 126 mg/dL ya sería un indicador de diabetes. Posteriormente se le ofrece una bebida acuosa con glucosa , de aproximadamente 75 gr, y se le vuelve a tomar una muestra de sangre. Una persona no diabética debería de obtener un resultado inferior a los 160 mg / dL, mientras que una persona diabética, puede superar fácilmente los 200 mg/dL.

- Por otra parte se mide también la presión arterial diastólica, es decir, la presión de las arterias cuando el corazn está descansando entre latido y latido, justo cuando el corazón se llena de sangre y ésta se oxigena. Una persona con una presión menor a 80 milímetros de mercurio (mmHg) se le considera sana. Una persona con más de 90 mmHg se le considera hipertensa y una persona con 120 mmHg o más, se le considera en estado de crisis de hipertensión, que puede causar convulsiones, dificultad para respirar o directamente la muerte.
- El espesor de la piel del tríceps en milímetros. El espesor del pliege cutáneo de esa parte puede predecir el porcentaje graso corporal. Para los hombres los valores normales son 2.5 mm y alrededor del 20% de grasa corporal. Para mujeres, 18 mm y al rededor del 30% de grasa.
- Otro parámetro que se estudia es la concentración de insulina a 2 horas vista. Una concentración superior a 150 mu U /ml.
- Otro parámetro que podemos encontrar es el índice de masa corporal que nos permite discriminar si el paciente está en riesgo de sobrepeso o infrapeso.
- Otro parámetro que encontramos es la función pedigrí de diabetes que mide si existen antecedentes de diabetes en los familiares y nos puede indicar si es un riesgo con influencia genética hereditaria o no.
- La edad.
- Y finalmente el resultado, es decir si sufre o no diabetes.

[1]:
```
# Cargamos las librearías de análisis exploratorio
import pandas as pd
import numpy as np
```

[2]:
```
# Cargamos las librerías de visualización
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[3]:
```
# Conjunto de dimensiones del conjunto de datos
variables = ["Numero de Embarazos","Concentracion Glucosa","Presion␣
↪Sanguinea","Grosor de piel","Concentracion de insulina a 2 horas␣
↪vista","Indice de masa corporal","Funcion de Diabetes","Edad","Class"]
```

[4]:
```
# Leemos el conjunto de datos
df = pd.read_csv("pima-indians-diabetes.csv",names=variables)
```

[5]:
```
# Exploramos los diez primeros casos
df.head(10)
```

[5]:

	Numero de Embarazos	Concentracion Glucosa	Presion Sanguinea	\
0	6	148	72	
1	1	85	66	
2	8	183	64	
3	1	89	66	

```
4                     0                     137                  40
5                     5                     116                  74
6                     3                      78                  50
7                    10                     115                   0
8                     2                     197                  70
9                     8                     125                  96

   Grosor de piel  Concentracion de insulina a 2 horas vista  \
0              35                                          0
1              29                                          0
2               0                                          0
3              23                                         94
4              35                                        168
5               0                                          0
6              32                                         88
7               0                                          0
8              45                                        543
9               0                                          0

   Indice de masa corporal  Funcion de Diabetes  Edad  Class
0                     33.6                 0.627    50      1
1                     26.6                 0.351    31      0
2                     23.3                 0.672    32      1
3                     28.1                 0.167    21      0
4                     43.1                 2.288    33      1
5                     25.6                 0.201    30      0
6                     31.0                 0.248    26      1
7                     35.3                 0.134    29      0
8                     30.5                 0.158    53      1
9                      0.0                 0.232    54      1
```

[6]:
```
# Comprobamos is hay valores perdidos
df.isnull().sum()
```

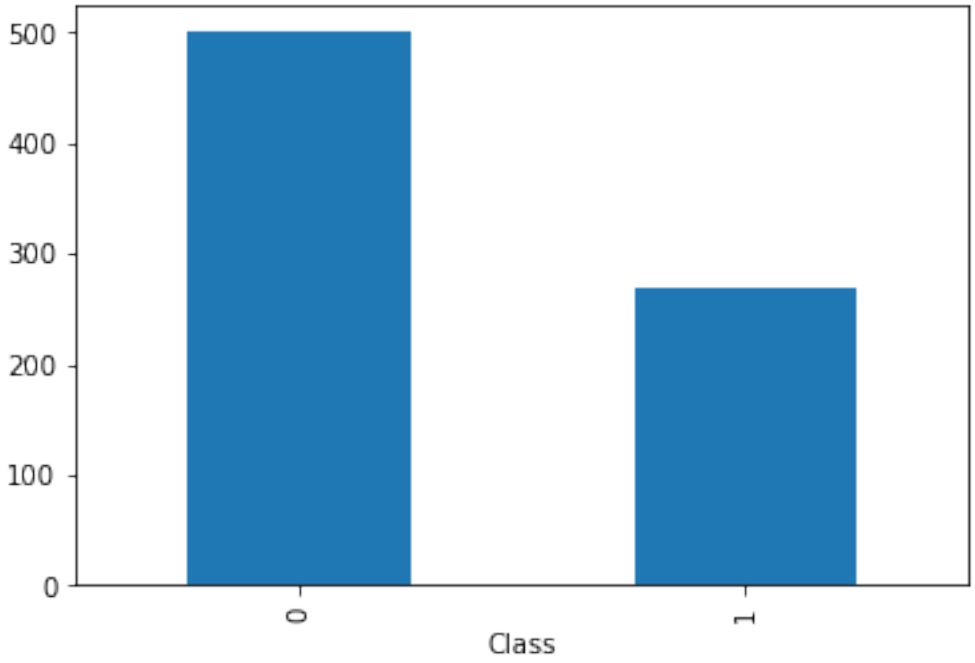
[6]:
```
Numero de Embarazos                       0
Concentracion Glucosa                     0
Presion Sanguinea                         0
Grosor de piel                            0
Concentracion de insulina a 2 horas vista    0
Indice de masa corporal                   0
Funcion de Diabetes                       0
Edad                                      0
Class                                     0
dtype: int64
```

[7]:
```
# Comprobamos los tipos de datos del dataset
df.dtypes
```

[7]:
```
Numero de Embarazos                          int64
Concentracion Glucosa                        int64
Presion Sanguinea                            int64
Grosor de piel                               int64
Concentracion de insulina a 2 horas vista    int64
Indice de masa corporal                    float64
Funcion de Diabetes                        float64
Edad                                         int64
Class                                        int64
dtype: object
```

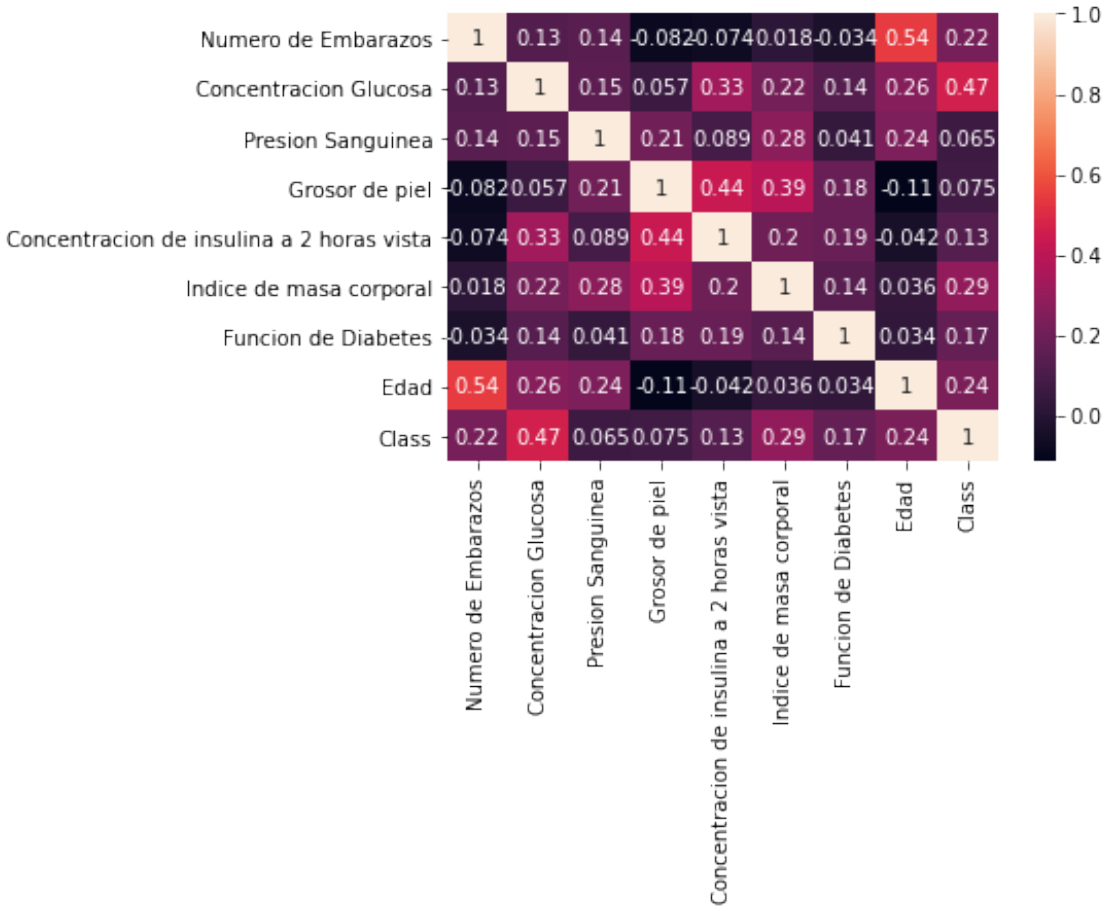
[8]:
```
# Comprobamos si el conjunto de datos está balanceado
df.groupby('Class').size()
df.groupby('Class').size().plot(kind="bar")
```

[8]:
```
<AxesSubplot:xlabel='Class'>
```



[9]:
```
# Comprobamos las correlaciones entre las dimesiones del dataset
correlaciones = df.corr()
sns.heatmap(correlaciones,annot=True)
```

[9]:
```
<AxesSubplot:>
```



[10]:
```
# Análisis descriptivo
df.describe().T
```

[10]:

	count	mean	std	\
Numero de Embarazos	768.0	3.845052	3.369578	
Concentracion Glucosa	768.0	120.894531	31.972618	
Presion Sanguinea	768.0	69.105469	19.355807	
Grosor de piel	768.0	20.536458	15.952218	
Concentracion de insulina a 2 horas vista	768.0	79.799479	115.244002	
Indice de masa corporal	768.0	31.992578	7.884160	
Funcion de Diabetes	768.0	0.471876	0.331329	
Edad	768.0	33.240885	11.760232	
Class	768.0	0.348958	0.476951	

	min	25%	50%	\
Numero de Embarazos	0.000	1.00000	3.0000	
Concentracion Glucosa	0.000	99.00000	117.0000	
Presion Sanguinea	0.000	62.00000	72.0000	

```
Grosor de piel                              0.000   0.00000   23.0000
Concentracion de insulina a 2 horas vista   0.000   0.00000   30.5000
Indice de masa corporal                     0.000  27.30000   32.0000
Funcion de Diabetes                         0.078   0.24375    0.3725
Edad                                       21.000  24.00000   29.0000
Class                                       0.000   0.00000    0.0000

                                                75%     max
Numero de Embarazos                         6.00000   17.00
Concentracion Glucosa                     140.25000  199.00
Presion Sanguinea                          80.00000  122.00
Grosor de piel                             32.00000   99.00
Concentracion de insulina a 2 horas vista 127.25000  846.00
Indice de masa corporal                    36.60000   67.10
Funcion de Diabetes                         0.62625    2.42
Edad                                       41.00000   81.00
Class                                       1.00000    1.00
```

1.3 Limpieza de datos

[11]:
```
# Reemplazamos los valores presion sanguinea 0 por la mediana. Si no tiene␣
 ↪presión estaría muerto
df['Presion Sanguinea'] = df['Presion Sanguinea'].
 ↪replace(to_replace=0,value=df['Presion Sanguinea'].median())
df['Indice de masa corporal'] = df['Indice de masa corporal'].
 ↪replace(to_replace=0,value=df['Indice de masa corporal'].median())

df['Concentracion de insulina a 2 horas vista'] = df['Concentracion de insulina␣
 ↪a 2 horas vista'].fillna(df['Concentracion de insulina a 2 horas vista'].
 ↪median())
df['Concentracion Glucosa'] = df['Concentracion Glucosa'].
 ↪replace(to_replace=0,value=df['Concentracion Glucosa'].median())
df['Grosor de piel'] = df['Grosor de piel'].fillna(df['Grosor de piel'].
 ↪median())
```

[12]:
```
df.isnull().sum()
```

[12]:
```
Numero de Embarazos                         0
Concentracion Glucosa                       0
Presion Sanguinea                           0
Grosor de piel                              0
Concentracion de insulina a 2 horas vista   0
Indice de masa corporal                     0
Funcion de Diabetes                         0
Edad                                        0
Class                                       0
dtype: int64
```

1.4 Aprendizaje automático

[21]:
```
# Cargamos las librerías de machine learning
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler as Scaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
```

1.4.1 Creamos el conjunto de entrenamiento y de test

[14]:
```
Xfeatures = df.iloc[:,0:8]
Ylabels = df['Class']

scaler = Scaler()

X = scaler.fit_transform(Xfeatures)
X = pd.DataFrame(X,columns=variables[0:8])

X_train,X_test,y_train,y_test = train_test_split(X,Ylabels,test_size=0.
↪2,random_state=42)
```

1.4.2 Regresión Lineal

[37]:
```
logit = LogisticRegression()
logit.fit(X_train,y_train)
print("Precisión Regresión lineal",logit.score(X_test,y_test))
```

```
Precisión Regresión lineal 0.7792207792207793
```

1.4.3 KNN

[19]:
```
knn = KNeighborsClassifier(n_neighbors=11)
knn.fit(X_train, y_train)
predicted_knn = knn.predict(X_test)
accuracy_knn = accuracy_score(y_test, predicted_knn)
print("Precisión Knn ", accuracy_knn)
```

```
Precisión Knn  0.7467532467532467
```

1.4.4 Perceptrón multicapa

[36]:
```
clf = MLPClassifier(solver='adam', alpha=1e-6, batch_size=2, activation='relu',␣
↪max_iter = 200, hidden_layer_sizes=(8, 2), random_state=1)
clf.fit(X_train, y_train)
predicted_clf = clf.predict(X_test)
accuracy_clf = accuracy_score(y_test, predicted_clf)
```

```
print("Precicsion MLP", accuracy_clf)
```

```
Precicsion MLP 0.7857142857142857
```

2 Explicación de los modelos

2.1 LIME

[38]:
```
import lime
import lime.lime_tabular
```

[39]:
```
explainer1 = lime.lime_tabular.LimeTabularExplainer(X_train.values,␣
↪feature_names=variables, class_names=['Sin Diabetes','Con Diabetes'],␣
↪discretize_continuous=True)
```

2.1.1 Explicación de 1 instancia del modelo de regresión lineal

[40]:
```
exp1 = explainer1.explain_instance(X_test.iloc[0],logit.
↪predict_proba,num_features=len(variables),top_labels=1)
```

[41]:
```
exp1.show_in_notebook(show_table=True, show_all=False)
```

```
<IPython.core.display.HTML object>
```

2.1.2 Explicación de 1 instancia del modelo KNN

[45]:
```
exp_knn = explainer1.explain_instance(X_test.iloc[0],
                                      knn.predict_proba,
                                      num_features=len(variables),
                                      top_labels=1)
exp_knn.show_in_notebook(show_table=True, show_all=False)
```

```
<IPython.core.display.HTML object>
```

2.1.3 Explicación de 1 instancia del modelo del perceptrón multicapa

[42]:
```
exp_multiperceptron = explainer1.explain_instance(X_test.iloc[0],clf.
↪predict_proba,num_features=len(variables),top_labels=1)
exp_multiperceptron.show_in_notebook(show_table=True, show_all=False)
```

```
<IPython.core.display.HTML object>
```

2.2 SHAP

[46]:
```
import shap
```

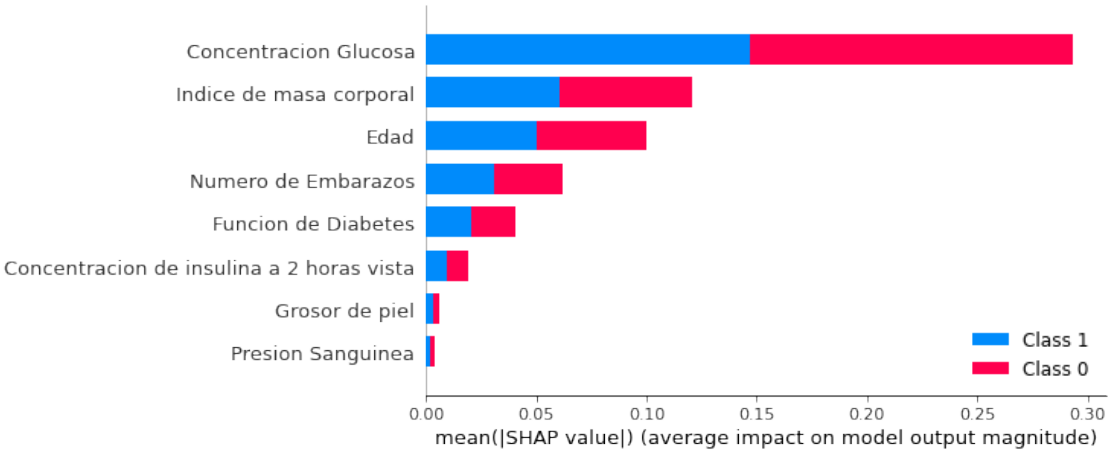
2.2.1 Explicación del modelo de regresión lineal

[47]:
```
shap.initjs()
explainer = shap.KernelExplainer(logit.predict_proba, X_train)
shap_values = explainer.shap_values(X_test)
shap.force_plot(explainer.expected_value[0], shap_values[0], X_test)
shap.summary_plot(shap_values,X_test)
```

```
<IPython.core.display.HTML object>
```

```
Using 614 background data samples could cause slower run times. Consider using
shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K
samples.
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=154.0),␣
↪HTML(value='')))
```

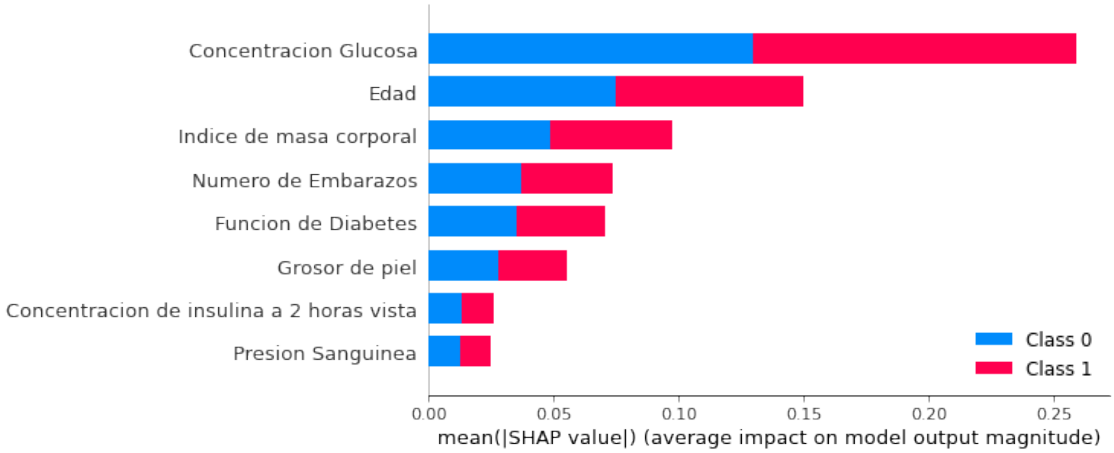


2.2.2 Explicación del modelo knn

[48]:
```
explainer_knn = shap.KernelExplainer(knn.predict_proba, X_train)
shap_values = explainer_knn.shap_values(X_test)
shap.force_plot(explainer_knn.expected_value[0], shap_values[0], X_test)
shap.summary_plot(shap_values,X_test)
```

```
Using 614 background data samples could cause slower run times. Consider using
shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K
samples.
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=154.0),␣
↪HTML(value='')))
```

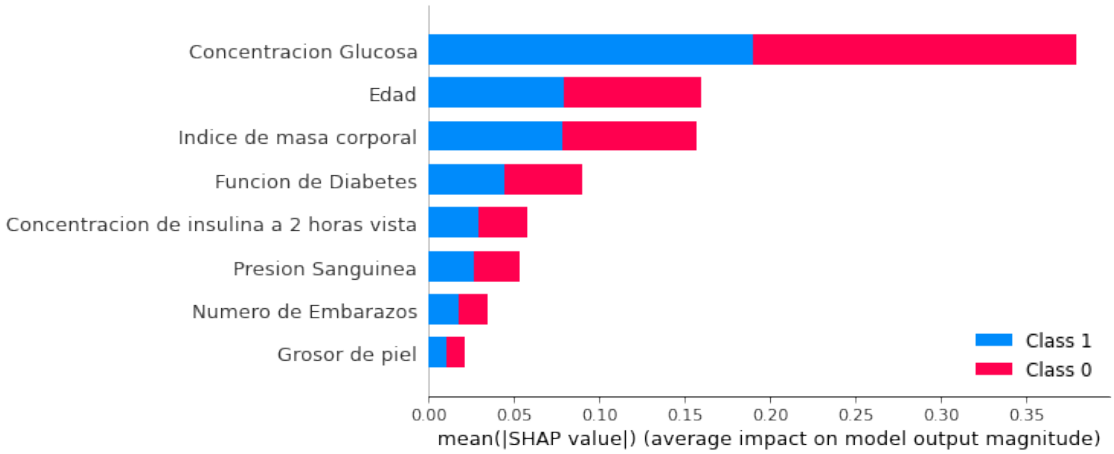


2.2.3 Explicación del modelo del perceptrón multicapa

[49]:
```
explainer_mlp = shap.KernelExplainer(clf.predict_proba, X_train)
shap_values = explainer_mlp.shap_values(X_test)
shap.force_plot(explainer_mlp.expected_value[0], shap_values[0], X_test)
shap.summary_plot(shap_values,X_test)
```

```
Using 614 background data samples could cause slower run times. Consider using
shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K
samples.
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=154.0),␣
↪HTML(value='')))
```



2.3 Adicional

2.3.1 Eli5

[50]:
```
import eli5
```

[51]:
```
eli5.show_weights(logit,feature_names=variables[0:8],target_names = ['Sin␣
↪Diabetes','Con Diabetes'])
```

[51]: <IPython.core.display.HTML object>

[54]:
```
eli5.show_weights(knn,feature_names=variables[0:8],target_names = ['Sin␣
↪Diabetes','Con Diabetes'])
```

[54]: <IPython.core.display.HTML object>

[55]:
```
eli5.show_weights(clf,feature_names=variables[0:8],target_names = ['Sin␣
↪Diabetes','Con Diabetes'])
```

[55]: <IPython.core.display.HTML object>

[ ]:
```

```