Actividad autocorregible. Conceptos generales de árboles y *random forest* para clasificación

Objetivos

En esta actividad aplicaremos los conceptos vistos sobre árboles de decisión y *random forest* para un problema de clasificación.

Descripción de la actividad

Para resolver las preguntas aquí planteadas, se deben solucionar los interrogantes del archivo **Act1.ipynb,** adjunto a esta actividad. Una vez solucionado en el cuaderno de Jupyter, debes contestar a las siguientes preguntas.

Preguntas

1. ¿Qué código permite crear los histogramas de todas las variables?

A.

Ptl.hist(x, density=True, bins=30).

Plt.show()

B.

plt.rcParams['figure.figsize'] = [20, 10];

df\_base.hist()

plt.show()

C.

plt.rcParams['figure.figsize'] = [20, 10];

ptl.hist(x, bins=number of bins).

plt.show()

D.

ax.hist(x, bins=[0,25,50,75,100]).

plt.show():

1. ¿Qué código permite reemplazar los ceros de los atributos por valores NAN de las columnas concentracionGlucosa, presionArterialSistolica, pliegueCutaneo, insulinaSerica e IMC? Realmente los ceros son valores faltantes:

A.

from numpy import nan

cols=['concentracionGlucosa','presionArterialSistolica','pliegueCutaneo','insulinaSerica','IMC']

df\_base[cols] = df\_base[cols].replace({'0':np.nan, 0:np.nan})

# Contar el número de nan por columna

print(df\_base.isnull().sum())

B.

df\_base.replace(0, np.nan, inplace=True)

C.

cols=['concentracionGlucosa','presionArterialSistolica','pliegueCutaneo', 'insulinaSerica','IMC', 'Diabetes']

df[cols] = df[cols].replace({'0':np.nan, 0:np.nan})

# Contar el número de nan por columna

print(df.isnull().sum())

D.

df[‘concentracionGlucosa’].replace([‘0’,’0’],’’,inplace=True)

df['presionArterialSistolica'].replace([‘0’,’0’],’’,inplace=True)

df[‘pliegueCutaneo'].replace([‘0’,’0’],’’,inplace=True)

df[‘insulinaSerica'].replace([‘0’,’0’],’’,inplace=True)

df[‘IMC'].replace([‘0’,’0’],’’,inplace=True)

1. ¿Cuántos valores NaN existen para la columna concentracionGlucosa?

A. 5.

B. 35.

C. 227.

D. 374.

1. ¿Qué código permite reemplazar los valores faltantes utilizando una interpolación lineal?

A.

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing\_values=np.nan, strategy='most\_frequent')

imputer = imputer.fit(df\_base\_missing\_mode)

df\_base\_missing\_mode = imputer.transform(df\_base\_missing\_mode)

B.

df\_base\_missing\_interpo=df\_base\_missing\_interpo.interpolate(method ='linear', limit\_direction ='forward')

display('Verificación de datos faltantes : {0}'.format(df\_base\_missing\_interpo.isnull().sum().max()))

C.

df\_base\_missing\_mean=df\_base\_missing\_mean.fillna(df\_base\_missing\_mean.mean())

display('Verificación de datos faltantes : {0}'.format(df\_base\_missing\_mean.isnull().sum().max()))

D.

df\_base\_missing\_dropnColumns=df\_base.copy()

perc = 25.0 # mayor al 25% de valores faltantes

min\_count = int(((100-perc)/100)\*df\_base\_missing\_dropnColumns.shape[0] + 1)

df\_base\_missing\_dropnColumns = df\_base\_missing\_dropnColumns.dropna( axis=1, thresh=min\_count)

1. ¿Qué correlación existe entre el IMC y el pliegueCutaneo?

A. Es menor de 0.1.

B. Está entre 0.1 y 0.2.

C. Está entre 0.5 y 0.7.

D. Es superior a 0.8.

1. Configura el árbol con la profundidad máxima de 15, el número máximo de atributos como sqrt, el número mínimo de muestras requeridas para estar en un nodo hoja a 1, y recuerda incluir la semilla aleatoria y la estrategia utilizada para elegir la división en cada nodo como best.

En este caso, el valor de la exactitud del modelo que entrena un árbol de decisión con el *dataset* que elimina las filas con valores faltantes:

A. Es menor de 0.1.

B. Está entre 0.3 y 0.5.

C. Está entre 0.6 y 0.8.

D. Es superior a 0.8.

1. ¿Cuál es el código que entrena un algoritmo *random forest* con 100 árboles, con entrenamiento de 2 árboles en paralelo y semilla aleatoria?

A.

clas\_rndforest = RandomForestClassifier(n\_estimators=2,n\_jobs=100, random\_state=semilla\_aleatoria)

clas\_rndforest.fit(train\_x,train\_y)

B.

clas\_rndforest = RandomForestClassifier(n\_estimators=100\*2, random\_state=semilla\_aleatoria)

clas\_rndforest.fit(train\_x,train\_y)

C. Los códigos A y B son correctos.

D.

clas\_rndforest = RandomForestClassifier(n\_estimators=100,n\_jobs=2, random\_state=semilla\_aleatoria)

clas\_rndforest.fit(train\_x,train\_y)

1. El valor de la exactitud del modelo que entrena el algoritmo *random forest* con el *dataset* que elimina las filas con valores faltantes es de:

A. Es menor de 0.1.

B. Está entre 0.6 y 0.8.

C. Está entre 0.3 y 0.5.

D. Es superior a 0.8.

1. Ten presente que después del proceso de imputación por interpolación quedarán algunos valores con NAN, deberás aplicar el proceso de imputación *default,* es decir, dropna( ) para tener datos limpios. Configura el árbol con la profundidad máxima de 15, el número máximo de atributos como sqrt, el número mínimo de muestras requeridas para estar en un nodo hoja a 1, y recuerda incluir la semilla aleatoria y la estrategia utilizada para elegir la división en cada nodo como best.

En este caso, los valores de precisión para las dos clases del modelo que se entrena con árboles de decisión y el *dataset* que guarda los valores imputados con la función de interpolación:

A. Para la clase 0, está entre 0.1 y 0.2, y para la clase 1, está entre 0.3 y 0.4.

B. Para la clase 0, está entre 0.3 y 0.4, y para la clase 1, está entre 0.3 y 0.4.

C. Para la clase 0, es superior a 0.8, y para la clase 1, está entre 0.6 y 0.7.

D. Para la clase 0, es superior a 0.8, y para la clase 1, está entre 0.3 y 0.4.

1. Los valores de *recall* para las dos clases del modelo que se entrena con *random* *forest* y el *dataset* que guarda los valores imputados con la función de interpolación es de:

A. Para la clase 0, está entre 0.6 y 0.7, y para la clase 1, está entre 0.3 y 0.4.

B. Para la clase 0, está entre 0.6 y 0.7, y para la clase 1, está entre 0.3 y 0.4.

C. Para la clase 0, es superior a 0.8, y para la clase 1, está entre 0.6 y 0.7.

D. Para la clase 0, es superior a 0.85, y para la clase 1, está entre 0.6 y 0.7.