


INSTITUTO TECNOLOGICO DE HERMOSILLO

Pablo Galaz Molina

S7C

Minería de Datos

Eduardo Antonio Hinojosa Palafox

Informe del Proyecto Final (Predicción de la Diabetes)

A Miércoles 14 de Diciembre del 2022.

2.- Introducción

- En el proyecto que se desarrollara a continuación se hará uso de un dataset para predecir si una persona cuenta con diabetes o no. La diabetes es una enfermedad que no se debe tomar a la ligera y es una de las que más muertes causan por año a nivel mundial.

- Aproximadamente 422 millones de personas en todo el mundo cuentan con algún tipo de diabetes, la diabetes es una enfermedad que no respeta edad, sexo, posición social, económica, etc.

- Casi 250,000 personas mueren cada año en el mundo por padecer diabetes, es decir el 20% de las muertes totales por año.

3.- Descripción del problema a desarrollar

- Lo que se busca desarrollar es predecir en base a medidas diagnosticas del paciente, verificar si este cuenta con algún tipo de diabetes que pueda poner en peligro o en inestabilidad su vida.

- La idea es que en base a datos de cada paciente poder hacer una predicción de si cuenta con algún tipo de diabetes.

4.- Descripción del conjunto de datos

- El conjunto de datos consta de lo siguiente:

- ❖ Pregnancies: Número de veces embarazadas.

- ❖ Glucose: concentración de glucosa en plasma a las 2 horas en una prueba de tolerancia oral a la glucosa.

- BloodPressure: presión arterial diastólica (mm Hg).
- SkinThickness: Grosor del pliegue cutáneo del tríceps (mm).
- Insulin: insulina sérica de 2 horas (mu U/ml).
- BMI o IMC: Índice de masa corporal (peso en kg/(altura en m)^2).
- DiabetesPedigreeFunction: función de pedigrí de diabetes.
- Age: Edad (años).
- Outcome: variable de clase (0 o 1).

Número de Instancias: 768

Número de atributos: 8 más clase

Para cada atributo: (todos con valores numéricos).

Number of times pregnant

Plasma glucose concentration a 2 hours in an oral glucose tolerance test

Diastolic blood pressure (mm Hg)

Triceps skin fold thickness (mm)

2-Hour serum insulin (mu U/ml)

Body mass index (weight in kg/(height in m)^2)

Diabetes pedigree function

Age (years)

Class variable (0 or 1)

5.- Descripción de la solución propuesta

- Haciendo uso de los datos proporcionados para cada paciente, se hará una predicción de acuerdo a los valores de los datos, para clasificar a los pacientes (variable de clase u Outcome) en respuestas binarias (0 y 1) y así separar a los que salieron con test positivo y negativo.

6.- Descripción del código utilizado

- Para empezar a trabajar vamos a importar las librerías que utilizaremos para llevar a cabo nuestro trabajo conforme vamos avanzado en el programa se van incluyendo más las librerías.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

- Se crea un objeto llamado data y cargamos el dataset.

```
data = pd.read_csv('diabetes.csv')
```

- Se comienza con el **Análisis Exploratorio de los Datos** para tener un mejor conocimiento de estos, primeramente con ayuda del método shape obtenemos nuestro número de filas y columnas.

```
filas, columnas = data.shape
print('Numero de filas: ', filas)
print('Numero de columnas: ', columnas)
```

```
Numero de filas:  768
Numero de columnas:  8
```

- Y así mismo como el shape hay muchos métodos para tener un mejor conocimiento de los datos como el info, describe, etc. Así mismo como una buena práctica verificar si hay valores nulos o valores duplicados.

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

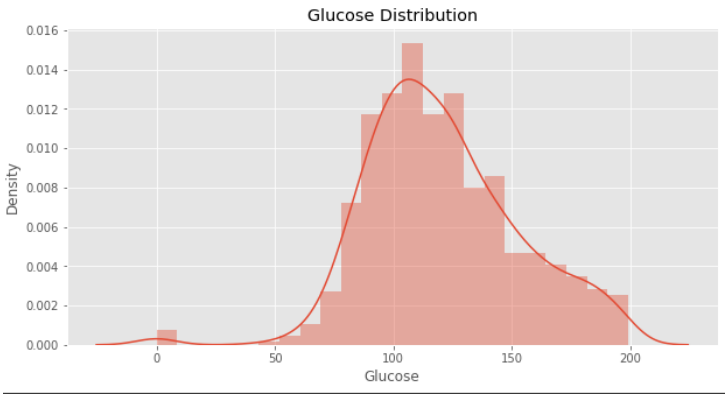
data.describe()

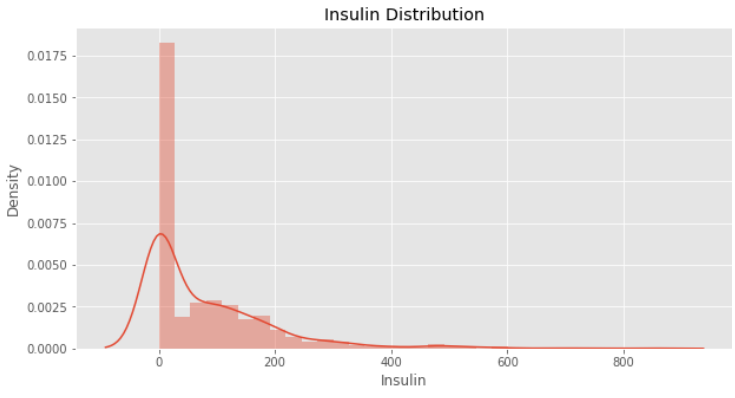
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

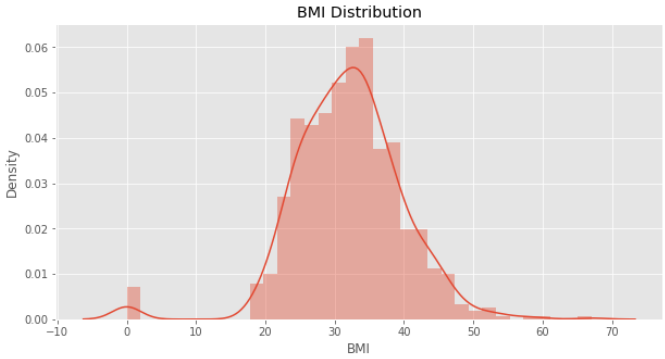
```
data.isnull().sum()

Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

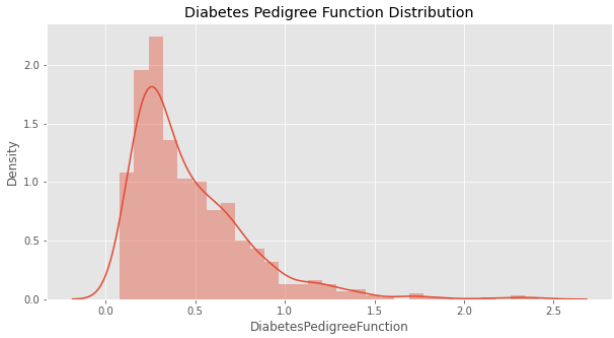
```
data.duplicated().sum()

0
```

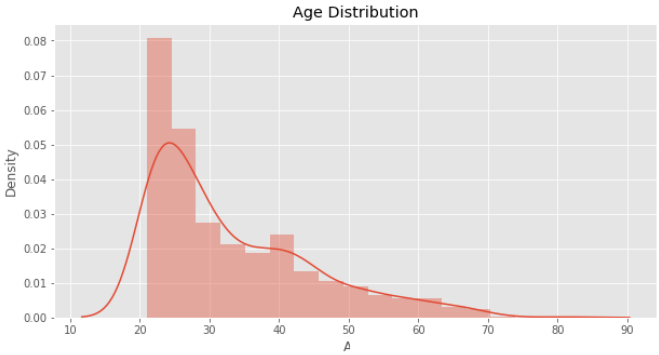
- A Continuación se pasa a la **Distribución de los datos** donde se procederá a graficar la distribución de los datos de cada una de las columnas con ayuda de las librerías plt y sns, fijando el tamaño de lo alto y ancho, su título, el dato de la columna a graficar.

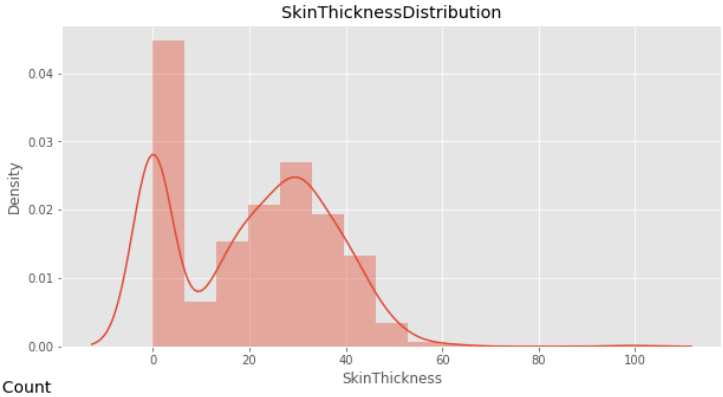


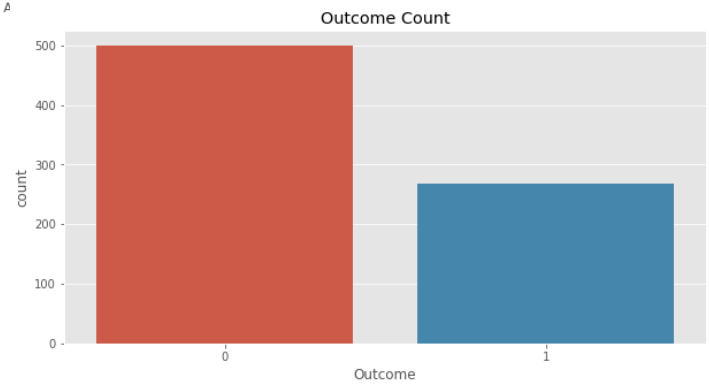












- Ahora pasamos al **pre procesamiento de los datos** donde se obtiene un conjunto de datos final. Establecemos un objeto llamado target y lo igualamos a la columna de Outcome del dataset, eliminando la primera columna que sería el titulo para que no se tome en cuenta.

```
target = data['Outcome']
data.drop(columns='Outcome', axis=1, inplace=True)
```

- Creamos un objeto llamado Scaler y lo igualamos a StandardScaler que es una función que nos sirve para estandarizar los datos. Y así mismo creamos otro objeto llamado newData igualando al data frame pasando como parámetro los datos de entrenamiento escalados del dataframe y las columnas.

```
scale = StandardScaler()
newData = pd.DataFrame(scale.fit_transform(data), columns=data.columns)
```

- Imprimimos el cabecero del nuevo dataframe (NewData) para ver cómo han cambiado los datos.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	0.468492	1.425995
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	-0.365061	-0.190672
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	0.604397	-0.105584
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	-0.920763	-1.041549
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	5.484909	-0.020496

- Así mismo cargamos la cabecera del target para ver como quedaron acomodados los datos.

```
target.head()
```

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

- Ahora se hace la **Prueba de Entrenamiento** al modelo.

```
xtrain, xtest, ytrain, ytest = train_test_split(newData, target, test_size=0.2, random_state=42)
```

- Haremos uso de un algoritmo de aprendizaje automático llamado SVC o en español clasificador de vectores de soporte que generalmente se usa en tareas de clasificación.

```
svc = SVC()

param = {
    'C': [i for i in range(1, 10)],
    'kernel': ['rbf', 'linear', 'poly']}

grid = GridSearchCV(svc, param, cv=5, scoring='neg_mean_squared_error')
grid.fit(xtrain, ytrain)
```

- A continuación ajustamos los parámetros de regresión lineal a los datos

```
SvcModel = grid.best_estimator_
SvcModel.fit(xtrain, ytrain)
```

```
SVC(C=2)
```

- Se evalúa las variables en los resultados.

```
svc_cv = cross_val_score(SvcModel, newData, target, cv=5)
```

- Para sacar la matriz de confusión creamos un objeto llamado model y lo igualamos al método de regresión logística, después se entrena el modelo y se predice el nuevo conjunto de datos.

- Se crea un objeto y se iguala a la matriz de confusión pasándole los dos parámetros de prueba, para posteriormente crear un nuevo dataframe con estos datos, renombrando las columnas y mostrando los resultados.

```
model = LogisticRegression()
model.fit(xtest,ytest)
y_pred = model.predict(xtest)
```

7.- Resultados

- Se imprimen los resultados:

```
print('Scores: ', svc_cv)
print('Mean: ', np.mean(svc_cv))
```

```
Scores:  [0.75974026 0.76623377 0.75324675 0.80392157 0.78431373]
Mean:  0.7734912146676851
```

- Se hace una función para imprimir los resultados.

```
def Main(modelName, model, cv):
  print(f'============ {modelName} ============')
  print('Model training error ---> ', np.mean(cv))
  print('Accuracy ---> ', accuracy_score(ytest, model.predict(xtest)))
  print('ConfusionMatix:')
  print(confusion_matrix(ytest, model.predict(xtest)))
  print()
  print(classification_report(ytest, model.predict(xtest)))
  print()
```

```
Main('SupportVectorMachine', SvcModel, svc_cv)
```

```
============ SupportVectorMachine ============
Model training error --->  0.7734912146676851
Accuracy --->  0.7207792207792207
ConfusionMatix:
[[81 18]
 [25 30]]

              precision    recall  f1-score   support

           0       0.76      0.82      0.79        99
           1       0.62      0.55      0.58        55

    accuracy                           0.72       154
   macro avg       0.69      0.68      0.69       154
weighted avg       0.71      0.72      0.72       154
```

- La matriz de confusión queda:

	No Diabetes	Diabetes
No Diabetes	87	12
Diabetes	19	36

8.- Conclusiones

- El algoritmo hace una predicción de valor real entre 0 y 1. Esto se transformó en una decisión binaria utilizando un límite de 0.448. Usando 576 instancias de entrenamiento, la sensibilidad y especificidad de su algoritmo fue del 76% en las 192 instancias restantes.

9.- Referencias

- https://www.kaggle.com/datasets/mathchi/diabetes-data-set
- https://www.paho.org/es/temas/diabetes

10.- Liga de GitHub

- ✓ https://github.com/Pablo-Galaz/Mineria-de-Datos