ML_CLASIFICACION_KNN_01	Modelo de Clasificación KNN para predicción de Diabetes	ML

En una consulta médica se dispone de una base de datos de 768 pacientes con la clasificación de aquellos que tienen 'diabetes'. (La diabetes es una enfermedad crónica en la que se produce un exceso de glucosa o azúcar en la sangre, que es debida a una disminución de la 'insulina' hormona producida por el páncreas).

La base de datos de pacientes contiene información de 8 atributos (X):

Número de embarazos	Presion en sangre (mm Hg)	Insulina 2-Hour insulin (mu U/ml)	Histórico (índice diabetes)
Glucosa (concentración a 2 horas)	Espesor de la piel (mm)	IMC (Índice de masa corporal. Kg/m^2)	Edad (años)

y una etiqueta (y) denominada "categoría" que indica si el paciente tiene diabetes (valor=1) o no (valor=0).

Con esta práctica se construye un modelo de aprendizaje supervisado basado en el algoritmo KNN (k-Nearest Neighbors), para permitir predecir la enfermedad de la 'diabetes' en nuevos pacientes.

Ejemplo de predicción:

Determinar, aplicando el modelo de aprendizaje supervisado, si un paciente con los siguientes 8 atributos tiene diabetes o no.

Atributos = [Embarazos: 1; Glucosa: 100; Presión en sangre: 70, Espesor de la piel:30;
Insulina: 90, IMC: 0.3; Histórico: 0.2; Edad: 45]

SOLUCIÓN

Importar las librerías necesarias para realizar la práctica.

```
# Importar librerías
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Lectura de la base de datos y consulta:

```
# Lectura de la base de datos "Base_diabetes.csv"
df = pd.read_csv("Base_diabetes.csv")

# Visualizar las primera 5 filas de la tabla de datos (DataFrame)
print(df.head())
```

```
   Embarazos  Glucosa    ...      Edad  Categoria
0          6      148    ...        50          1
1          1       85    ...        31          0
2          8      183    ...        32          1
3          1       89    ...        21          0
4          0      137    ...        33          1

[5 rows x 9 columns]
```

```
# Ver el tamaño de la tabla de datos
print(df.shape)
```

```
(768, 9)
```

```
# La base de datos tiene 768 registros, con 8 atributos y 1 etiqueta
# Atributos: Embarazos, Glucosa, Presión_sangre, Espesor_piel, Insulina, BMI,
Histórico,Edad
# Etiqueta: Categoría (1-> Tiene Diabetes     0-> No tiene diabetes)
```

Generar los datos X e y.

```
# Crear los arrays para X e y
X = df.drop('Categoria',axis=1).values
y = df['Categoria'].values
```

Dividirlos para entrenamiento y test:

```
# Dividir el conjunto de datos en entrenamiento y test
# Utilizar el método 'train_test_split' de la librería sklearn
from sklearn.model_selection import train_test_split

# Considerar como test un 40 % (test_size=0.4) del total del conjunto de datos
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.4,random_state=42, stratify=y)
```

Definir el clasificador basado en KNN y definir arrays de precisión (accuracy):

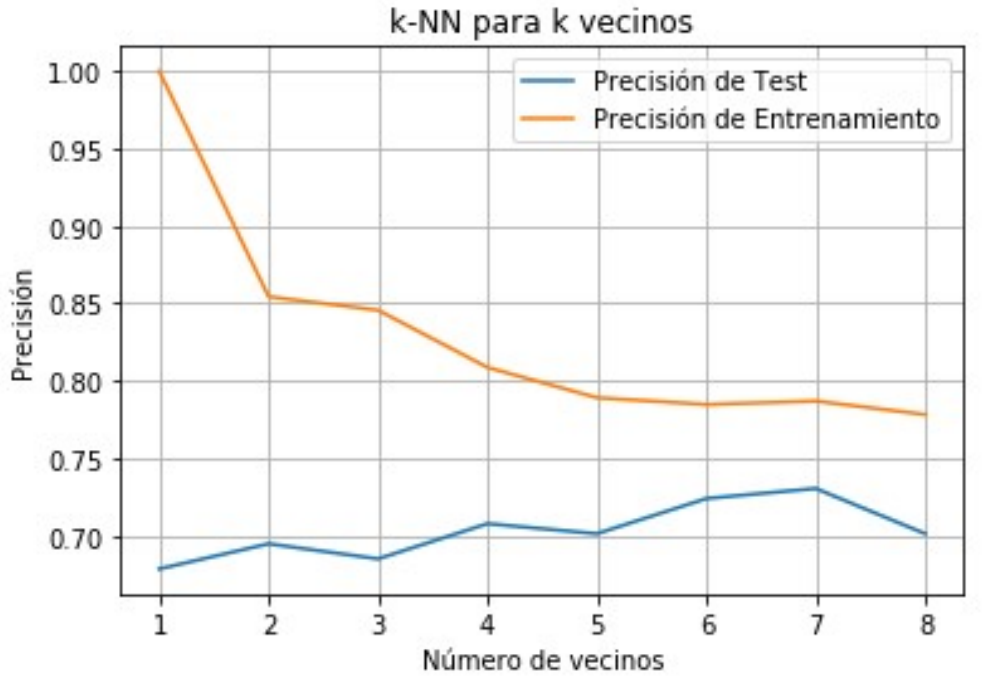
```
# Crear el clasificador basado en KNN (k-Nearest Neighbors)
from sklearn.neighbors import KNeighborsClassifier

# Crear los arrays donde registramos las precisiones para entrenamiento y test
neighbors = np.arange(1,9)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
```

Iterar para diferentes valores de 'k':

```
# Realizar el cálculo con diferentes valores de 'k'
# para identificar el valor de k que da mejores resultados

for i,k in enumerate(neighbors):
    # Configurar el clasificador KNN con 'k' vecinos (neighbors)
    knn = KNeighborsClassifier(n_neighbors=k)

    # Ajustar el modelo a los datos de entrenamiento
    knn.fit(X_train, y_train)

    # Registrar las precisiones para los datos de entrenamiento
    train_accuracy[i] = knn.score(X_train, y_train)

    # Registrar las precisiones para los datos de test
    test_accuracy[i] = knn.score(X_test, y_test)
```

Dibujar la gráfica de precisión de entrenamiento y test:

```
# Visualizar el gráfico de la precisión de entrenamiento y test
plt.title('k-NN para ''k'' vecinos')
plt.plot(neighbors, test_accuracy, label='Precisión de Test')
plt.plot(neighbors, train_accuracy, label='Precisión de Entrenamiento')
plt.legend()
plt.xlabel('Número de vecinos')
plt.ylabel('Precisión')
plt.grid()
plt.show()
```



Calcular la matriz de confusión, para un valor de 'k' igual a 7 (máxima precisión).

```
# La máxima precisión del ajuste con los valores de test se consigue con k=7
# Configurar el clasificador KNN con '7' vecinos (neighbors)
knn = KNeighborsClassifier(n_neighbors=7)

# Ajustar el modelo a los datos de entrenamiento
knn.fit(X_train,y_train)

# ERRORES/METRICAS
# Calcular la matriz de confusion (confusion matrix)
# Para ver sobre el conjunto de test los resultados

# Importar el método
from sklearn.metrics import confusion_matrix

# Predecir etiquetas sobre el conjunto de test
y_pred = knn.predict(X_test)

# Obtener la matriz de confusión
cm=confusion_matrix(y_test,y_pred)
print(cm)
print()
print("Verdaderos positivos (VP):",cm[1][1])
print("Verdaderos negativos (VN):",cm[0][0])
print("Falsos positivos (FP):",cm[0][1])
print("Falsos negativos (FN):",cm[1][0])
```

```
[[165  36]
 [ 47  60]]

Verdaderos positivos (VP): 60
Verdaderos negativos (VN): 165
Falsos positivos (FP): 36
Falsos negativos (FN): 47
```

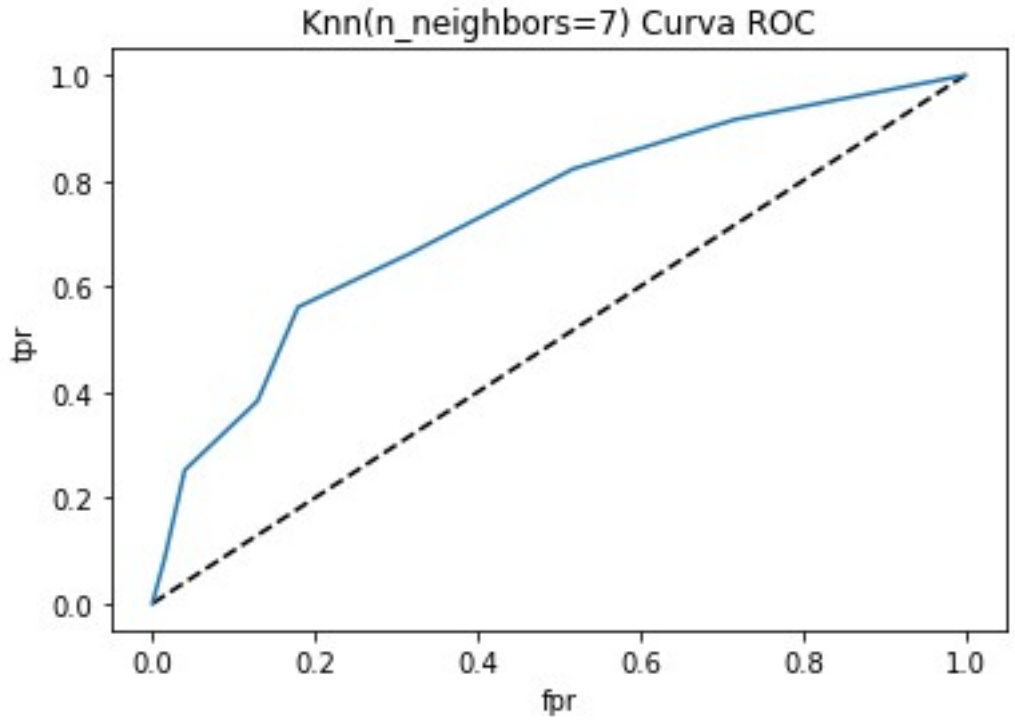
Presentar un informe de clasificación con diferentes métricas:

```
# Informe de clasificación
from sklearn.metrics import classification_report

print()
print(" Informe de clasificación (sobre datos de test")
print(classification_report(y_test,y_pred))
```

```
Informe de clasificación (sobre datos de test
             precision    recall  f1-score   support

          0       0.78      0.82      0.80       201
          1       0.62      0.56      0.59       107

avg / total       0.73      0.73      0.73       308
```

Y realizar la curva característica (curva ROC):

```
# Curva característica (Curva ROC)
# Curva de falsos positivos (FP) frente a verdaderos positivos (VP)
from sklearn.metrics import roc_curve

# Predecir probabilidades sobre el conjunto de test
y_pred_proba = knn.predict_proba(X_test)[:,1]

# Calcular los ratios de falsos positivos (fpr) y verdadero positivo (tpr)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Dibujar la curva ROC
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=7) Curva ROC')
plt.show()
```




```
# Calcular el área por encima de la curva ROC
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_proba)
```

```
Área :  0.7345050448691124
```

Predecir para un nuevo paciente con atributos:[Embarazos: 1; Glucosa: 100; Presión en sangre: 70, Espesor de la piel:30; Insulina: 90, IMC: 0.3; Histórico: 0.2; Edad: 45]

```
# Predecir para un nuevo paciente
# Atributos:Atributos = [Embarazos: 1; Glucosa: 100; Presión en sangre: 70,
Espesor de la piel:30;
# Insulina: 90, IMC:  0.3; Histórico: 0.2; Edad: 45]

y_pred_nuevo = knn.predict([[1,100,70,30,90,0.3,0.2,45]])

if y_pred_nuevo==1:
    print("El paciente tiene diabetes")
else:
    print("El paciente no tiene diabetes")


El paciente no tiene diabetes
```