Actividad 2
Memoria

Inteligencia Artificial

ALEJANDRO GARCÍA GARCÍA / JORGE GARCÍA MARTÍN

Índice

1. Introducción

La memoria de un proyecto es aquella en la que se recopila toda la información acerca del proyecto, tanto planificación, como bibliografías, como explicaciones, etc.

Para este proyecto, hemos dividido la memoria en:

- Planificación: Resumimos brevemente cómo hemos planificado el proyecto para llevarlo a cabo.
  En este apartado también entrarían los conocimientos previos, explicamos los conocimientos previos al desarrollo del proyecto y que hemos tenido que hacer para desarrollarlo correctamente.

- Desarrollo: En esta sección repasamos todo el código que hemos implementado para que el programa realice lo pedido.

- Pruebas: Se enseñan capturas con las pruebas que hemos realizado para comprobar que el programa realmente funciona

- Bibliografía: Enseñamos los enlaces de los sitios donde hemos encontrado la información para realizar el proyecto.

2. Planificación

Puesto que no teníamos todos los conocimientos previos para realizar todo el ejercicio correctamente, primero investigamos un poco sobre el lenguaje de programación R, intentando programar el programa en ese lenguaje.

Al final nos dimos cuenta de que nos resulta más sencillo implementarlo en Python, puesto que ya partíamos de una base y sabíamos manejarlo mejor que R.

Como conocimientos previos, partimos, como ya hemos comentado anteriormente, de programar en Python algunos programas, sabiendo ya las librerías que hay que usar y entendiendo mejor el código y funciones que tenemos que realizar para este trabajo, y por eso decidimos programarlo en Python, ya que, si lo programábamos en R, al no tener conocimientos previos a realizar la práctica, nos supondría más difícil realizarla, y gastando además mucho más tiempo.

En cuanto a la planificación, comenzamos buscando los métodos necesarios para implementar el programa, intentando, asimismo, comprender todo el ejercicio. Una vez comprendido, procedemos a realizar el código y a realizar las pruebas unas cuantas veces para asegurarnos de que todo funcionaba correctamente.
Tras la comprobación de todo el código, por último, realizamos la memoria del proyecto para plasmar en ella todo lo necesario acerca de esta práctica.

3. Desarrollo

Como ya hemos comentado anteriormente en el apartado de planificación, el desarrollo del código lo realizamos en Python.
En este apartado explicaremos resumidamente el código que hemos utilizado para el correcto funcionamiento de este:

- Para empezar, importamos todas las librerías necesarias para que el código funcionase.

```
import matplotlib
import numpy as np
import pandas as p
import matplotlib.pyplot as plot
import seaborn as sb

from sklearn import svm, metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LogisticRegression
```

- Implementamos dos arrays vacíos y realizamos dos métodos para llenarlas

```
matplotlib.use("TkAgg")

precisiones_array = []
kfold_validation_array = []

kfold = KFold(n_splits=10)


def add_acc_to_array(Y_test, prediction):
    accurancy = metrics.accuracy_score(y_true=Y_test, y_pred=prediction)
    precision = metrics.precision_score(y_true=Y_test, y_pred=prediction)
    recall = metrics.recall_score(y_true=Y_test, y_pred=prediction)
    f1 = metrics.f1_score(y_true=Y_test, y_pred=prediction)
    data = [accurancy, precision, recall, f1]
    precisiones_array.append(data)


def add_kfold_to_array(modelo, data, outcome):
    kfold_res = cross_val_score(modelo, data, outcome, cv=kfold)
    kfold_validation_array.append(kfold_res.mean())
```

- Luego realizaremos el main, que contendrá el resto del código. En esta captura aparece el código donde se sustituyen los 0 por los NaN y más tarde se eliminan las filas que contienen NaN de entre las columnas Glucose, BloodPressure, SkinThickness, Insulin y BMI, puesto que son datos imposibles, excluyendo columnas como Outcome.

```
diabetes[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]] = diabetes[
    ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]].replace(0, np.nan)
diabetes = diabetes.dropna()
print(">Filas después de preprocesado: " + str(len(diabetes.index)))
print(diabetes.head())
print("#" * 84)
```

- Análisis exploratorio: Se realiza el gráfico con el Outcome para comprobar cuantos salen con diabetes y cuantos no, y a continuación se crea un pairplot para crear los gráficos cruzando los datos del Outcome y las distintas columnas.

```
count_an = sb.countplot(x="Outcome", data=diabetes)
count_an.set_xticklabels(labels=["No", "Yes"])
count_an.set(xlabel="Outcome", ylabel="Personas")

plot.title("OUTCOME")
plot.show()

sb.pairplot(data=diabetes, hue="Outcome", diag_kind="kde")
plot.show()

outcome = diabetes["Outcome"]
data = diabetes[diabetes.columns[:8]]
```

- A continuación, dividimos en train y test el conjunto de datos, como secciones de datos numéricos y categóricos.

```
train, test = train_test_split(diabetes, test_size=0.33, random_state=42, stratify=diabetes["Outcome"])
```

- Una vez tenemos los conjuntos train con X(categorías), e Y(datos numéricos), los introducimos en los distintos modelos de clasificación.

```
# SVM Lineal
modelo = svm.SVC(kernel="linear", random_state=122)
modelo.fit(X_train, Y_train)
prediction = modelo.predict(X_test)
add_acc_to_array(Y_test, prediction)
add_kfold_to_array(modelo, data, outcome)

# Regresion Logistica
modelo = LogisticRegression(max_iter=600, random_state=122)
modelo.fit(X_train, Y_train)
prediction = modelo.predict(X_test)
add_acc_to_array(Y_test, prediction)
add_kfold_to_array(modelo, data, outcome)

# Decision Tree
modelo = DecisionTreeClassifier(random_state=122)
modelo.fit(X_train, Y_train)
prediction = modelo.predict(X_test)
add_acc_to_array(Y_test, prediction)
add_kfold_to_array(modelo, data, outcome)

# K Neighbors
modelo = KNeighborsClassifier(n_neighbors=10)
modelo.fit(X_train, Y_train)
prediction = modelo.predict(X_test)
add_acc_to_array(Y_test, prediction)
add_kfold_to_array(modelo, data, outcome)
```

```
# Naive Bayes
modelo = BernoulliNB(binarize=True)
modelo.fit(X_train, Y_train)
prediction = modelo.predict(X_test)
add_acc_to_array(Y_test, prediction)
add_kfold_to_array(modelo, data, outcome)
```

- A continuación, programamos el código para que el programa realice la matriz de correlación, con ‘coolwarm’ como color y contando las columnas pertinentes.

```
color = sb.color_palette("coolwarm", as_cmap=True)
sb.heatmap(diabetes[diabetes.columns[:8]].corr(), annot=True, cmap=color)
fig = plot.gcf()
fig.set_size_inches(20, 14)
plot.title("MATRIZ DE CORRELACIÓN")
plot.show()
```

- Más tarde se crea un conjunto de datos (DataFrame) que reunirá los datos de la validación mediante Kfold obtenidos del array que se ha ido rellenando cada vez que se usaba un método de clasificación

```
modelos = ["SVM", "Regresion Logistica", "Decision Tree", "K-Nearest", "Naive Bayes"]
tabla_kfold = p.DataFrame(index=modelos, columns=["KFold (K = 10)"], data=kfold_validation_array)
p.options.display.max_columns = 100
p.options.display.float_format = "{:,.7f}".format
print("#" * 84)
print(tabla_kfold)
print("#" * 84)
sb.barplot(tabla_kfold.T)
plot.title("KFold Cross Validation (K = 10)")
plot.show()
```

- Creamos otro conjunto de datos y lo mostramos por terminal en forma de tabla para poder comparar los datos exactos.

```
nombres_precisiones_calculadas = ["Accurancy", "Precision", "Recall", "F1"]
tabla_precisiones = p.DataFrame(index=nombres_precisiones_calculadas)
p.options.display.float_format = "{:,.7f}".format
for data, nombre_col in zip(precisiones_array, modelos):
    tabla_precisiones[nombre_col] = p.array(data=data)
print("#" * 84)
print(tabla_precisiones)
print("#" * 84)
```

- Por último, creamos e imprimimos un gráfico con los datos de precisiones para mostrar la precisión de cada método de manera más visual.

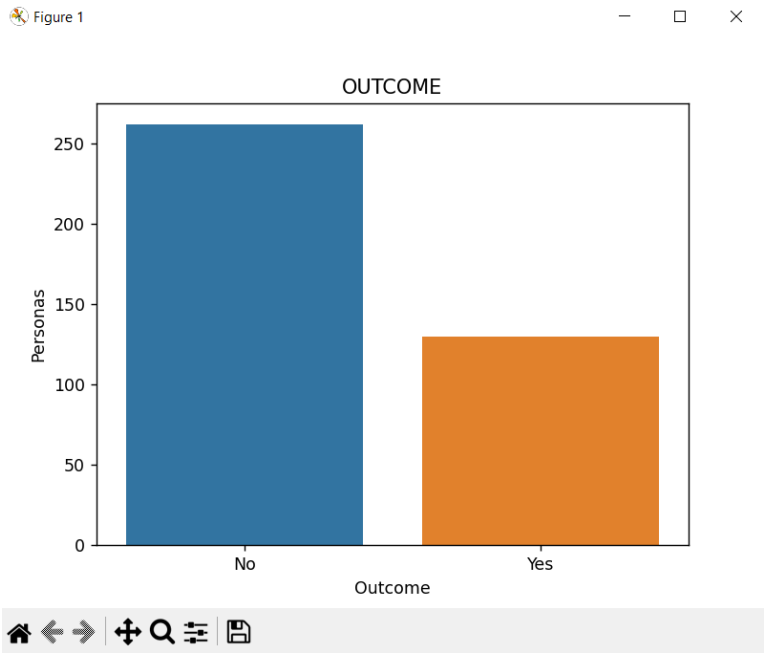
```
box = p.DataFrame(precisiones_array, index=modelos)

sb.boxenplot(box.T)
plot.title("Precisiones métodos de clasificación")
plot.show()
```

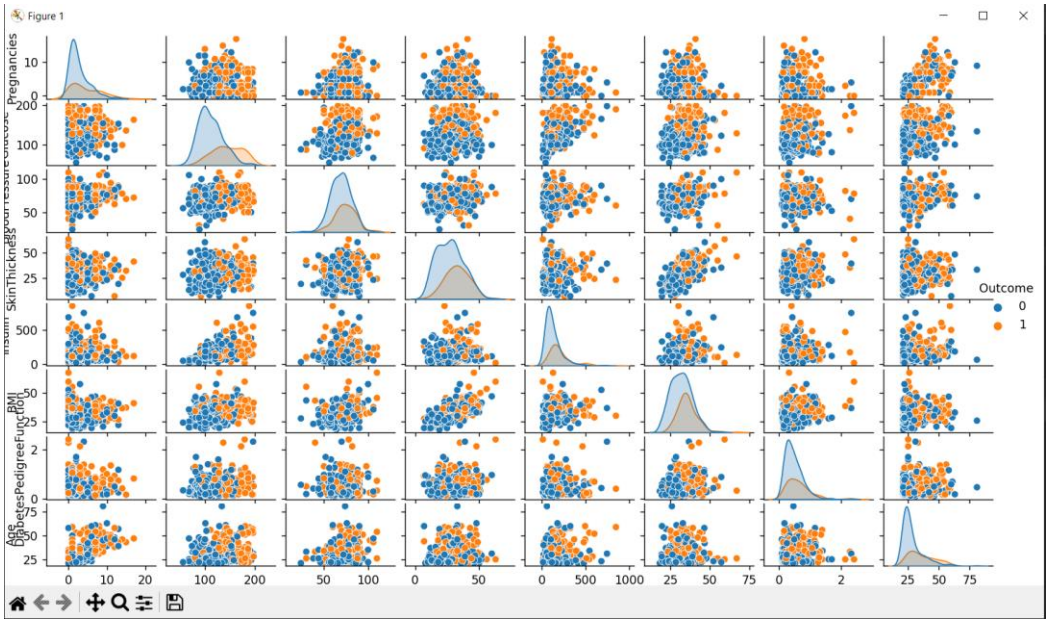
Este código programado en Python sería lo necesario para que el programa funcionase correctamente, aun así en el siguiente apartado aparecerá una prueba realizada para comprobar que todo el código funciona y se muestra correctamente el material gráfico.

4. Pruebas

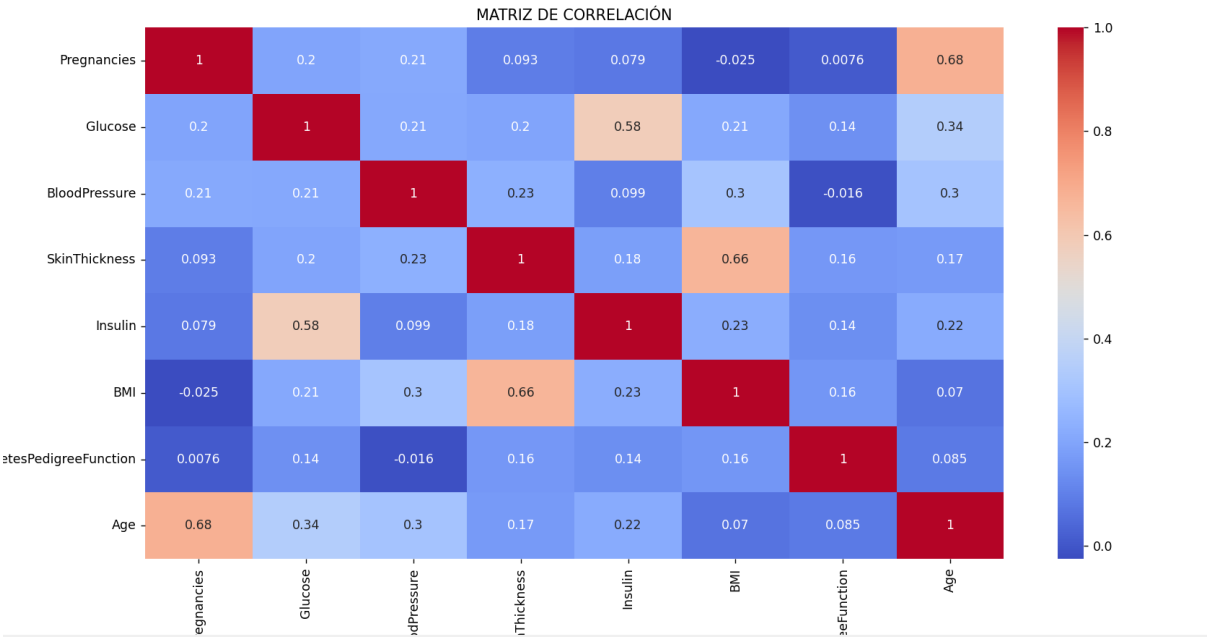
Para la comprobación del ejercicio, realizamos varias pruebas observando los errores que nos iban saliendo y así poder solucionarlos, hasta comprobar que, todo funciona correctamente, cómo vamos a mostrar a continuación:

- Gráfico del Outcome:

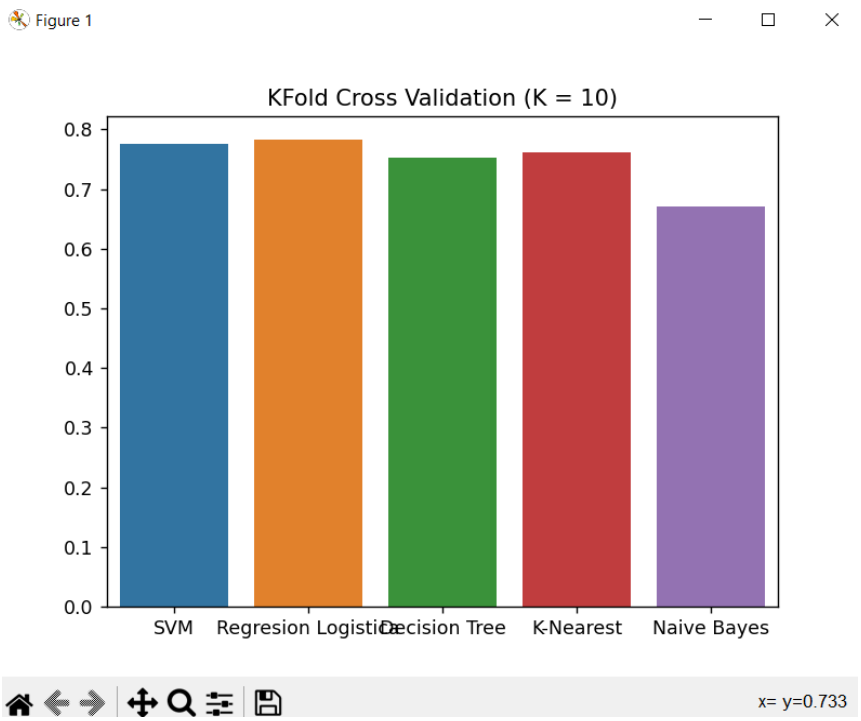




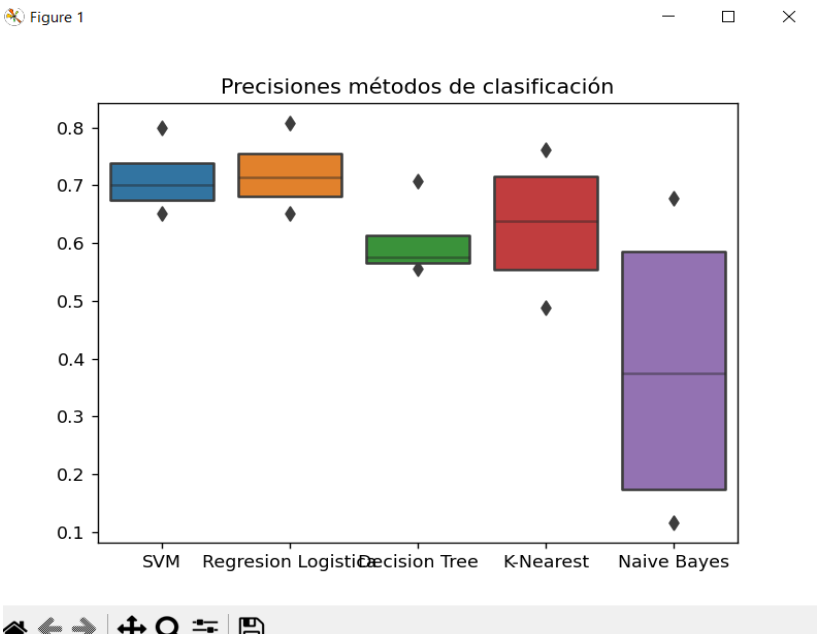
- Matriz de correlación:



- Gráfico del KFold:



- Gráfico de precisiones de los métodos de clasificación:



- Las diferentes tablas que tienen que imprimirse por consola con todos los datos:

```
>Filas antes de preprocesado: 768
>Filas después de preprocesado: 392
    Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
3             1     89.0           66.0  ...                     0.167   21        0
4             0    137.0           40.0  ...                     2.288   33        1
6             3     78.0           50.0  ...                     0.248   26        1
8             2    197.0           70.0  ...                     0.158   53        1
13            1    189.0           60.0  ...                     0.398   59        1

[5 rows x 9 columns]
################################################################################
################################################################################
                    KFold (K = 10)
SVM                      0.7752564
Regresion Logistica      0.7829487
Decision Tree            0.7528846
K-Nearest                0.7624359
Naive Bayes              0.6709615
################################################################################
################################################################################
                SVM  Regresion Logistica  Decision Tree  K-Nearest  \
Accurancy 0.8000000           0.8076923      0.7076923  0.7615385
Precision 0.7179487           0.7368421      0.5555556  0.7000000
Recall    0.6511628           0.6511628      0.5813953  0.4883721
F1        0.6829268           0.6913580      0.5681818  0.5753425
```

	Naive Bayes
Accurancy	0.6769231
Precision	0.5555556
Recall	0.1162791
F1	0.1923077

5. Conclusión

Tras ejecutar todo el código y analizar tanto los gráficos mostrados como las precisiones impresas por la consola, podemos concluir que el método que nos proporciona una mayor precisión es el de Regresión Logística.

Regresión Logística (*Logistic Regression*) muestra un valor de *Accurancy* de ~0.8077 y muestra una gran fidelidad en la validación cruzada (*Cross Validation*) usando el método *KFold* con un valor de ~0.7829, ambos siendo los valores más altos de entre los métodos usados.

6. Bibliografía

- https://scikit-learn.org/stable/getting_started.html - Documentación de *sklearn*
- https://stackoverflow.com/ - Resolución de dudas variadas
- https://www.pythontutorial.net/tkinter/tkinter-matplotlib/ - Ayuda con Matplotlib y Tkinter
- https://www.w3schools.com/python/pandas - Como apoyo a la hora de usar Pandas y los DataFrames
- https://tryolabs.com/blog/2017/03/16/pandas-seaborn-a-guide-to-handle-visualize-data-elegantly - Combinación de Pandas y Seaborn