Modelo predictivo de la flor de Iris paso a paso

El proceso de creación de un modelo predictivo implica varias etapas, pero en resumen, consiste en seleccionar y preparar los datos, elegir un algoritmo de aprendizaje automático, entrenar el modelo con los datos a nuestra disposición y luego evaluar su rendimiento para establecer cuáles de los algoritmos usados es el más efectivo.

Definir a principio de código todas las versiones instaladas de las librerías que se usarán puede ser útil para asegurarse de que se estén utilizando las versiones más actualizadas y compatibles con otros paquetes que se vayan a utilizar.

```
#Importación de las librerias de Python, comúnmente utilizadas en análisis de datos,
#aprendizaje automático y visualización.
# Python version
#sys, es una biblioteca incorporada de Python que proporciona acceso a varias variables y
#funciones utilizadas para interactuar con el sistema operativo.
import sys
print('Python: {}'.format(sys.version))
# scipy, es una biblioteca de Python utilizada para matemáticas, ciencia e ingeniería.
import scipy
print('scipy: {}'.format(scipy.__version__))
# numpy, es una biblioteca de Python utilizada para trabajar con matrices y arreglos numéricos.
import numpy
print('numpy: {}'.format(numpy.__version__))
# matplotlib, es una biblioteca de Python utilizada para graficar y visualizar datos.
import matplotlib
print('matplotlib: {}'.format(matplotlib.__version__))
# pandas, es una biblioteca de Python utilizada para el análisis de datos y
#la manipulación de estructuras de datos.
import pandas
print('pandas: {}'.format(pandas.__version__))
# scikit-learn, es una biblioteca de Python utilizada para el aprendizaje automático
#y la minería de datos.
import sklearn
print('sklearn: {}'.format(sklearn.__version__))
```

Figura 1. Importación de las librerías y discusión.

```
from pandas.plotting import scatter_matrix #se utiliza para crear una matriz
# de gráficos de dispersión para
#explorar la relación entre las variables.
import matplotlib.pyplot as plt #es una librería de visualización de datos que se utiliz
# para crear gráficos y visualizaciones.
from sklearn import model_selection #se utiliza para dividir los datos en
#conjuntos de entrenamiento y prueba,
# y también para realizar validación cruzada.
from sklearn.metrics import classification_report #en las siguientes tres
#lineas de codigo se utilizan
#librerias para evaluar el rendimiento de los modelos de clasificación.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression #a continuación
#siguen una serie de algoritmos de clasificación que se utilizan para
#construir modelos de aprendizaje automático.
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Figura 2. Importación de las librerías y discusión.

El siguiente código de Python lee el conjunto de datos sobre las flores del género Iris desde

el archivo de texto "iris.flower.dataset.txt".

path = "C:/Users/aless/Downloads/iris.flower.dataset.txt"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pandas.read_csv(path, names=names)

La librería "pandas" se utiliza para leer y manipular datos en formato tabular y se utiliza para leer el archivo de texto en un objeto de datos llamado "dataset". El código utiliza la función "read_csv" de la librería "pandas" para leer el archivo de texto y los nombres de las columnas se especifican con el argumento "names=names".

A partir de aquí se realiza un análisis exploratorio de datos, un proceso de investigación de datos con el objetivo de resumir sus principales características. Esta técnica se utiliza para comprender mejor los datos antes de aplicar técnicas de modelado o inferencia.

print(dataset.shape)
Es una instrucción en Python que devuelve la forma (shape) de un conjunto de datos (dataset) en términos de su tamaño y dimensiones. La salida de esta instrucción es una tupla que contiene dos valores: el número de fila y el número de columnas del conjunto de datos. En este caso tenemos un conjunto de datos con 150 filas y 5 columnas y la salida de la instrucción "print(dataset.shape)" será (150, 5).

print(dataset.head(20))
Es una instrucción en Python que imprime las primeras 20 filas del conjunto de datos (dataset) en la consola. Imprimir las primeras filas del conjunto de datos es una buena práctica para visualizar su contenido y comprobar si se ha cargado correctamente. También puede ser útil para identificar problemas con los datos, como valores faltantes o datos incorrectos, y para determinar la estructura y formato de los datos.

```
(150, 5)
    sepal-length  sepal-width  petal-length  petal-width        class
0            5.1          3.5           1.4          0.2  Iris-setosa
1            4.9          3.0           1.4          0.2  Iris-setosa
2            4.7          3.2           1.3          0.2  Iris-setosa
3            4.6          3.1           1.5          0.2  Iris-setosa
4            5.0          3.6           1.4          0.2  Iris-setosa
5            5.4          3.9           1.7          0.4  Iris-setosa
6            4.6          3.4           1.4          0.3  Iris-setosa
7            5.0          3.4           1.5          0.2  Iris-setosa
8            4.4          2.9           1.4          0.2  Iris-setosa
9            4.9          3.1           1.5          0.1  Iris-setosa
10           5.4          3.7           1.5          0.2  Iris-setosa
11           4.8          3.4           1.6          0.2  Iris-setosa
12           4.8          3.0           1.4          0.1  Iris-setosa
13           4.3          3.0           1.1          0.1  Iris-setosa
14           5.8          4.0           1.2          0.2  Iris-setosa
15           5.7          4.4           1.5          0.4  Iris-setosa
16           5.4          3.9           1.3          0.4  Iris-setosa
17           5.1          3.5           1.4          0.3  Iris-setosa
18           5.7          3.8           1.7          0.3  Iris-setosa
19           5.1          3.8           1.5          0.3  Iris-setosa
```

Figura 4. Visualización en la consola de la ejecución de las líneas de código correspondiente a '**print(dataset.shape)**' y '**print(dataset.head(20))**'.

print(dataset.describe())

Es una instrucción en Python que imprime un resumen estadístico del conjunto de datos (dataset) en la consola. Estas estadísticas incluyen el número de valores no nulos, la media, la desviación estándar, los valores mínimo y máximo, así como los percentiles 25%, 50% y 75% de los datos. El resumen estadístico que se imprime es útil para entender la distribución de los datos, identificar valores atípicos
(outliers) y valores faltantes, y para determinar si es necesario aplicar alguna transformación o preprocesamiento a los datos antes de utilizarlos en un modelo de aprendizaje automático.

```
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

Figura 5. Visualización en la consola del resumen estadístico.

print(dataset.groupby('class').size())

"print(dataset.groupby('class').size())" es una instrucción en Python que imprime el recuento (size) de cada clase en un conjunto de datos (dataset). La función "groupby()" es un método que se utiliza para agrupar los datos según los valores de una o varias columnas. En este

caso, se está agrupando el conjunto de datos por la columna 'class'. El resultado de la agrupación se presenta como una tabla que muestra el número de instancias en cada clase del conjunto de datos. Esta información es importante para entender la distribución de las clases en el conjunto de datos y puede ser útil para tomar decisiones sobre cómo dividir el conjunto de datos en conjuntos de entrenamiento y prueba, así como para evaluar la precisión de un modelo de aprendizaje automático.

```
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

Figura 6. Visualización en la consola de la tabla con el número de instancias en cada clase del conjunto de datos.

Para completar el análisis exploratorio de datos y tener una mejor comprensión del conjunto de datos se procede a la visualización de estos.

1. Gráficos para entender mejor el comportamiento de cada variable.
2. Gráficos multivariante para mejor entender la relación entre variables.

1. Gráficos univariante

dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()

Este código genera un gráfico de diagrama de caja (box plot) para cada columna de un conjunto de datos (dataset). Los argumentos de la función plot() especifican que se deben crear subgráficos para cada una de las cuatro columnas del conjunto de datos (layout=(2,2)), y que no se debe compartir el eje X ni el eje Y entre los subgráficos

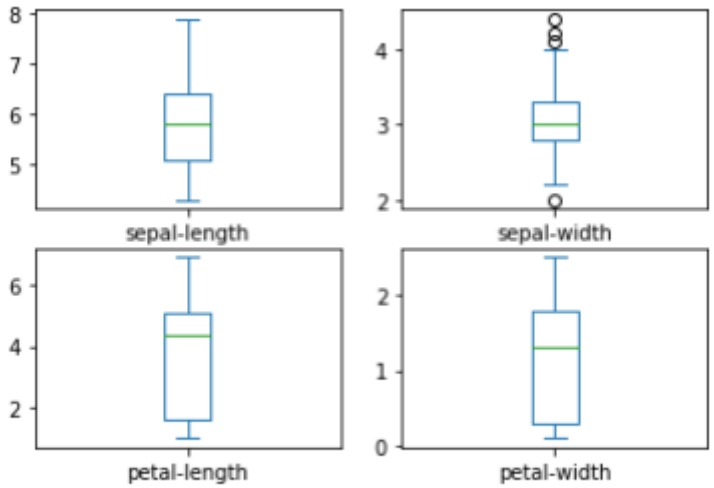


Figura 8. Boxplot

dataset.hist()
plt.show()

Este código genera un histograma para cada columna de un conjunto de datos (dataset).

La función hist() de matplotlib permite visualizar la distribución de frecuencia de los valores de una variable.

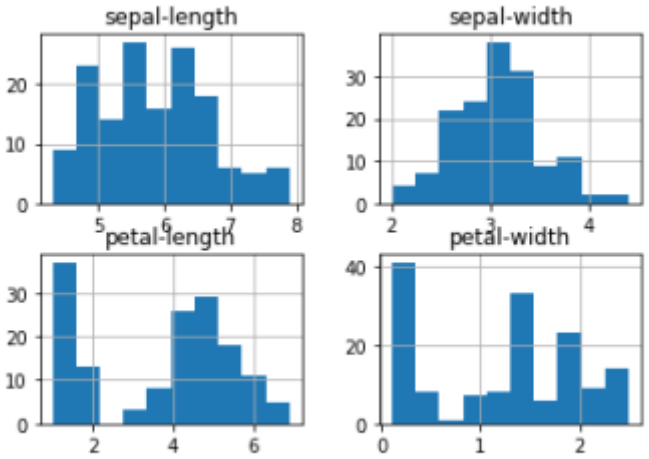


Figura 9. Histograma.

2. Gráficos multivariante

scatter_matrix(dataset)

plt.show()

Este código genera una matriz de gráficos de dispersión (scatter plot matrix) para el conjunto de datos (dataset). La función scatter_matrix() de pandas permite visualizar la relación entre cada par de variables en el conjunto de datos.

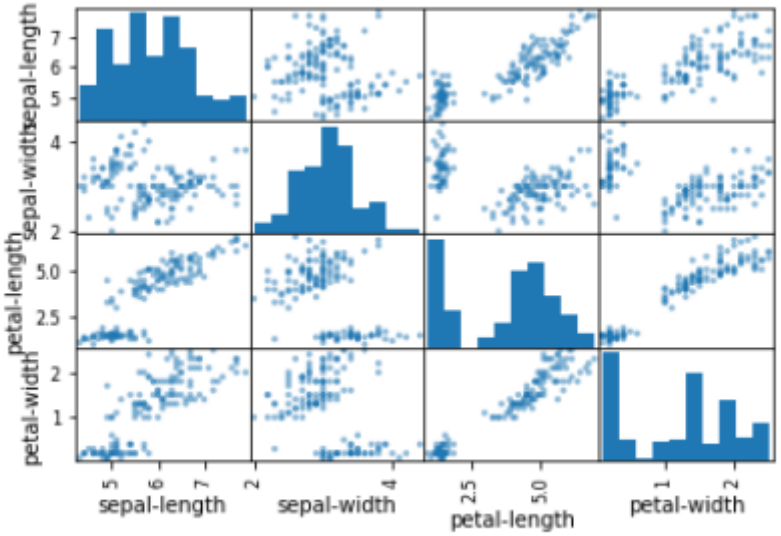


Figura 10. Matriz de gráficos de dispersión.

A continuación se procede a evaluar una serie de algoritmos con los que se entrenará el modelo. El proceso de evaluación consistirá en:

1. Crear un conjunto de datos de validación
2. Validación cruzada

3. Construir 5 modelos diferentes para predecir la especie de Iris a partir de medidas de flores
4. Seleccionar el mejor modelo

1. Crear un conjunto de datos de validación.

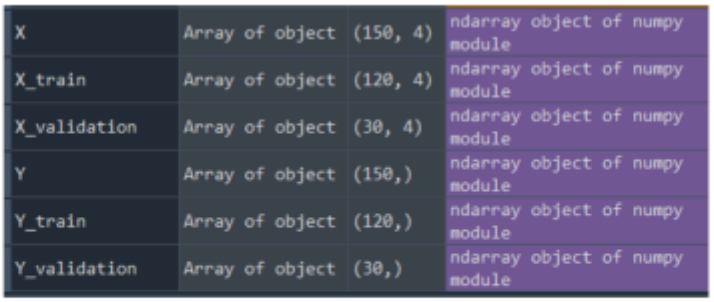
array = dataset.values
Convierte el conjunto de datos (dataset) en un array de NumPy, es decir, una estructura de datos bidimensional que contiene los valores del conjunto de datos. Cada fila del array representa una observación (registro) del conjunto de datos y cada columna representa una variable.
dataset.columns #devuelve los nombres de las columnas del dataset.
dataset.index #devuelve los nombres de los índices del dataset.
dataset.values #devuelve los valores del dataset en formato de array.

```
Out[15]:
array([[5.1, 3.5, 1.4, 0.2, 'Iris-setosa'],
       [4.9, 3.0, 1.4, 0.2, 'Iris-setosa'],
       [4.7, 3.2, 1.3, 0.2, 'Iris-setosa'],
       [4.6, 3.1, 1.5, 0.2, 'Iris-setosa'],
       [5.0, 3.6, 1.4, 0.2, 'Iris-setosa'],
       [5.4, 3.9, 1.7, 0.4, 'Iris-setosa'],
       [4.6, 3.4, 1.4, 0.3, 'Iris-setosa'],
       [5.0, 3.4, 1.5, 0.2, 'Iris-setosa'],
       [4.4, 2.9, 1.4, 0.2, 'Iris-setosa'],
       [4.9, 3.1, 1.5, 0.1, 'Iris-setosa'],
       [5.4, 3.7, 1.5, 0.2, 'Iris-setosa'],
       [4.8, 3.4, 1.6, 0.2, 'Iris-setosa'],
       [4.8, 3.0, 1.4, 0.1, 'Iris-setosa'],
       [4.3, 3.0, 1.1, 0.1, 'Iris-setosa'],
       [5.8, 4.0, 1.2, 0.2, 'Iris-setosa'],
       [5.7, 4.4, 1.5, 0.4, 'Iris-setosa'],
       [5.4, 3.9, 1.3, 0.4, 'Iris-setosa'],
       [5.1, 3.5, 1.4, 0.3, 'Iris-setosa'],
       [5.7, 3.8, 1.7, 0.3, 'Iris-setosa'],
       [5.1, 3.8, 1.5, 0.3, 'Iris-setosa'],
       [5.4, 3.4, 1.7, 0.2, 'Iris-setosa'],
       [5.1, 3.7, 1.5, 0.4, 'Iris-setosa'],
       [4.6, 3.6, 1.0, 0.2, 'Iris-setosa'],
       [5.1, 3.3, 1.7, 0.5, 'Iris-setosa'],
       [4.8, 3.4, 1.9, 0.2, 'Iris-setosa'],
```

Figura 11. Array

X = array[:,0:4] #Esta línea selecciona las columnas del 0 al 3 (4 no inclusive) del array para ser usadas como variables independientes en el modelo.
Y = array[:,4] #Esta línea selecciona la columna 4 del array para ser usada como variable dependiente en el modelo.
validation_size = 0.20 #Esta línea define el tamaño de la muestra de validación como el 20% del dataset.
seed = 1 #Esta línea establece la semilla para el generador de números aleatorios, lo que asegura que la división de los datos en entrenamiento y validación se realice de la misma manera cada vez que se ejecute el código.
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size=validation_size, random_state=seed, stratify='array-like'
Esta ultima línea divide los datos en un conjunto de entrenamiento y un conjunto de validación usando la función train_test_split de la librería scikit-learn. Los argumentos son las variables independientes (X), la variable dependiente (Y), el tamaño de la muestra de validación (validation_size), la semilla (seed), y stratify='array-like', lo que significa que se desea una división estratificada de los datos, es decir, mantener la proporción de cada clase de la variable dependiente tanto en el conjunto de entrenamiento como en el de validación.

Los resultados de la división son cuatro arrays: X_train y Y_train son el conjunto de entrenamiento, mientras que X_validation y Y_validation son el conjunto de validación.



X	Array of object	(150, 4)	ndarray object of numpy module
X_train	Array of object	(120, 4)	ndarray object of numpy module
X_validation	Array of object	(30, 4)	ndarray object of numpy module
Y	Array of object	(150,)	ndarray object of numpy module
Y_train	Array of object	(120,)	ndarray object of numpy module
Y_validation	Array of object	(30,)	ndarray object of numpy module

Figura 12. Visualización de las variables creadas.
Array X-train, con el 80% de los datos e X_validation con el 20% restante.

2. Validación cruzada y test de hardness.

El "test harness" se refiere al proceso de evaluar el rendimiento de un modelo de aprendizaje automático en un conjunto de datos mediante el uso de una métrica de evaluación específica. En este caso, se utiliza una validación cruzada de 10 pliegues para estimar la precisión del modelo, lo que significa que se dividirá el conjunto de datos en 10 partes, se entrenará en 9 y se probará en 1, y se repetirá para todas las combinaciones posibles de divisiones de entrenamiento-prueba. La métrica de evaluación utilizada es la precisión (‘accuracy’).

scoring = 'accuracy'

scoring	str	8	accuracy

Figura 13. Resultado de la validación cruzada.

3. Construir 5 modelos diferentes para predecir la especie de Iris a partir de medidas de flores

```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
#evaluate each model in turn
results = []
names = []

for name, model in models:
        kfold = model_selection.KFold(n_splits=10)
        cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
```

msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print(msg)

En este paso se construyen los modelos para predecir especies a partir de medidas de flores. Para ello se evalúan 6 algoritmos diferentes: Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbors, Decision Tree Classifier, Gaussian Naive Bayes y Support Vector Machine.
Luego, utiliza la técnica de validación cruzada (cross-validation) para evaluar cada modelo en la lista, utilizando el conjunto de datos de entrenamiento (X_train, Y_train) y un número de divisiones (10). El resultado de cada modelo se almacena en una lista de resultados y se muestra en la consola el promedio y la desviación estándar de los resultados obtenidos por cada modelo.

```
Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/
linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LR: 0.966667 (0.040825)
LDA: 0.983333 (0.033333)
KNN: 0.983333 (0.033333)
CART: 0.958333 (0.055902)
NB: 0.958333 (0.041667)
SVM: 0.975000 (0.038188)
```

Figura 14. Error y R cuadro de cada modelo.

4. Selección del mejor modelo mediante comparación entre modelos

fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()

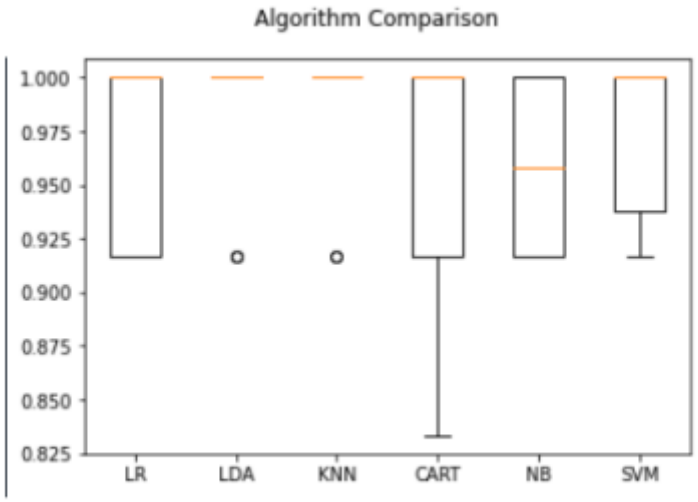


Figura 15. Comparación de los algoritmos.

Los algoritmos LDA y KNN resultan los modelos más precisos. Para tener una idea de la verdadera precisión de los modelos evaluamos uno de ellos con el conjunto de datos de validación. En este caso se opta por el algoritmo KNN.

knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation) #esto sería el test (con el 20% de los datos).

Finalmente, para tener una idea del rendimiento del modelo se procede a ejecutar las tres siguientes líneas:
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))

La primera línea, accuracy_score(Y_validation, predictions), calcula la precisión del modelo en el conjunto de datos de validación. Toma dos argumentos, Y_validation que contiene las etiquetas verdaderas y predictions que contiene las etiquetas predichas por el modelo. La precisión se define como el número de predicciones correctas dividido por el número total de predicciones.
La segunda línea, confusion_matrix(Y_validation, predictions), crea una matriz de confusión para el modelo. La matriz de confusión muestra el número de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. La matriz se utiliza para evaluar la calidad del modelo y puede ser útil para determinar qué clases están siendo mal clasificadas por el modelo.
La tercera línea, classification_report(Y_validation, predictions), genera un informe de clasificación que proporciona información sobre la precisión, la recuperación y la puntuación F1 del modelo para cada clase. La precisión mide la proporción de instancias clasificadas como positivas que son realmente positivas. La recuperación mide la proporción de instancias positivas que son correctamente clasificadas como positivas. La puntuación F1 es una medida de la precisión y la recuperación y se calcula como la media armónica de ambas. Este informe puede ser útil para determinar qué clases están siendo clasificadas correctamente y cuáles necesitan mejorar.

```
0.9333333333333333
[[ 8  0  0]
 [ 0  9  2]
 [ 0  0 11]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       1.00      0.82      0.90        11
 Iris-virginica       0.85      1.00      0.92        11

       accuracy                           0.93        30
      macro avg       0.95      0.94      0.94        30
   weighted avg       0.94      0.93      0.93        30
```

Figura 16. Precisión, matriz de confusión e informe de clasificación.