Quinta parte: documentación

https://github.com/koalagordokbron/cell_net_evaluacion_modelos

Vamos a explicar cómo funciona el modelo elegido, Random Forest.

Recomendaciones

Antes de empezar a trabajar sobre el notebook es recomendable tener las siguientes librerías instaladas:

```
import os
import pandas as pd
import random
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns

from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model

from tensorflow.keras.utils import to_categorical
from tensorflow.random import set_seed

from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier

from fast_ml.model_development import train_valid_test_split

from IPython.display import Image

from sklearn.metrics import (
    accuracy_score, log_loss, confusion_matrix, ConfusionMatrixDisplay,
    classification_report, roc_curve, auc
)
```

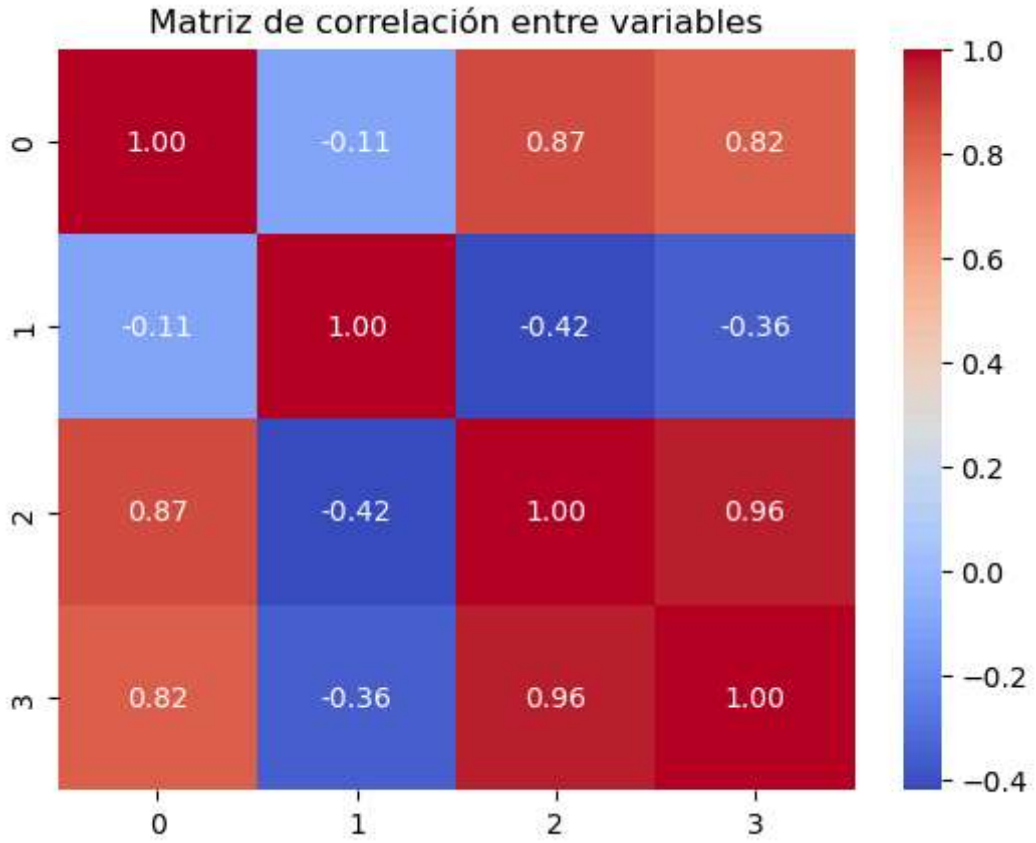
1. Datos

Los datos los hemos obtenidos de la siguiente URL:
https://raw.githubusercontent.com/Xachap/cell-network-example/main/cells.csv

Consiste en un dataset con 150 registros de células, recoge 4 medidas de las dimensiones de las mismas en micrometros, y en base a estas mediciones las clasifica en 'Benigno', 'Normal' y 'Maligno'.

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Normal
1	4.9	3.0	1.4	0.2	Normal
2	4.7	3.2	1.3	0.2	Normal
3	4.6	3.1	1.5	0.2	Normal
4	5.0	3.6	1.4	0.2	Normal
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Maligno
146	6.3	2.5	5.0	1.9	Maligno
147	6.5	3.0	5.2	2.0	Maligno
148	6.2	3.4	5.4	2.3	Maligno
149	5.9	3.0	5.1	1.8	Maligno

En la siguiente imagen podemos ver la correlación de las variables:



Para empezar a trabajar con estos datos, debemos dividirlos en dos grupos:

- X: son las variables que usaremos como entrada.
- y: es la variable objetivo.

```
# Dividimos el conjunto de datos en variables de entrada (X) y salida (Y)
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
```

También podemos dividirlos usando el modulo `pandas`, que personalmente prefiero.

Lo siguiente que haremos será dividirlos en:

- Train.
- Test.

Así podemos entrenar el modelo y hacer una prueba realista con datos que nunca ha visto este con el mismo conjunto de datos.

2. Modelo

El modelo que hemos elegido es Random Forest, el cual genera varios árboles de decisión para hacer las predicciones.

Hemos definido una función que devuelve el modelo según los parámetros seleccionados:

```
def random_forest_model(estimators=100, seed=SEED, depth=15):
    model = RandomForestClassifier(
        n_estimators=estimators,
        random_state=seed,
        max_depth=depth,
        n_jobs=-1
    )

    return model
```

✓ 0.0s

```
# creamos el modelo
model_rf = random_forest_model(350, SEED, 1)
```

- n_estimators: el número de árboles de decisión que queremos crear.
- random_state: para asegurar la reproducibilidad del modelo.
- max_depth: cómo de grandes van a ser los árboles (profundidad).
- n_jobs: núcleos del procesador a usar.

2.1 Validación cruzada

Antes de entrenar el modelo realizamos validación cruzada con StratifiedKFold:

```
15 # validación cruzada
16 skf = StratifiedKFold(
17     n_splits=5, # 80/20
18     shuffle=True,
19     random_state=SEED
20 )
21
22 cv_scores = cross_val_score(model_rf, X_train_rf, y_train_rf, cv=skf, scoring='recall_macro')
```

Al hacer validación cruzada, en este caso con 5 splits, creamos un conjunto de validación dentro de los datos de entrenamiento.

Esto lo hacemos para hacer una especie de "simulación" antes de hacer el test real y ver una aproximación de los resultados que vamos a obtener.

```
---Resultados de la validación cruzada---
Puntuación media: 0.95
Desviación estándar: 0.031180478223116 2
```

3. Entrenamiento

Entrenamos el modelo con los datos de entrenamiento que hemos segmentado anteriormente:

```
28 # entranimiento del modelo
29 model_rf.fit(X_train_rf, y_train_rf)
```

Antes de hacer una evaluación sobre los resultados obtenidos, hacemos una predicción:

```
31 # predicción
32 y_pred_rf = model_rf.predict(X_test_rf)
```

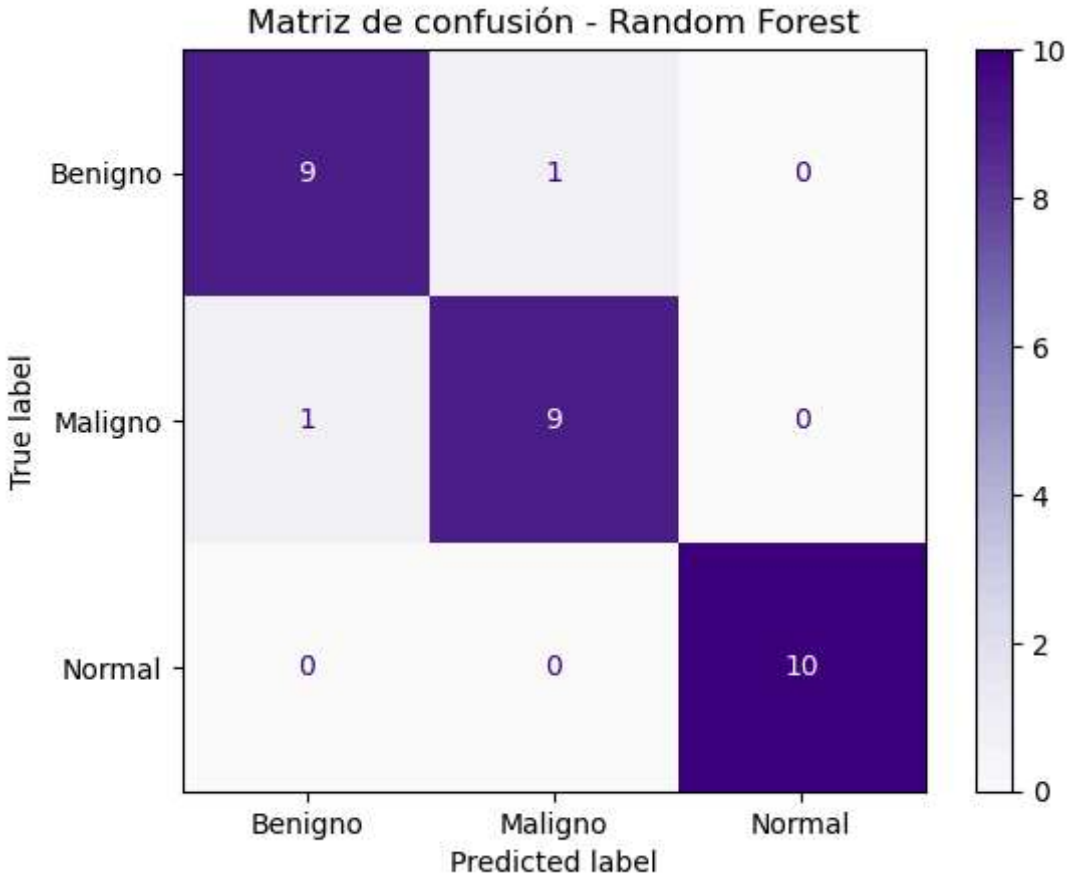
Utilizamos el conjunto de entrada de Test.

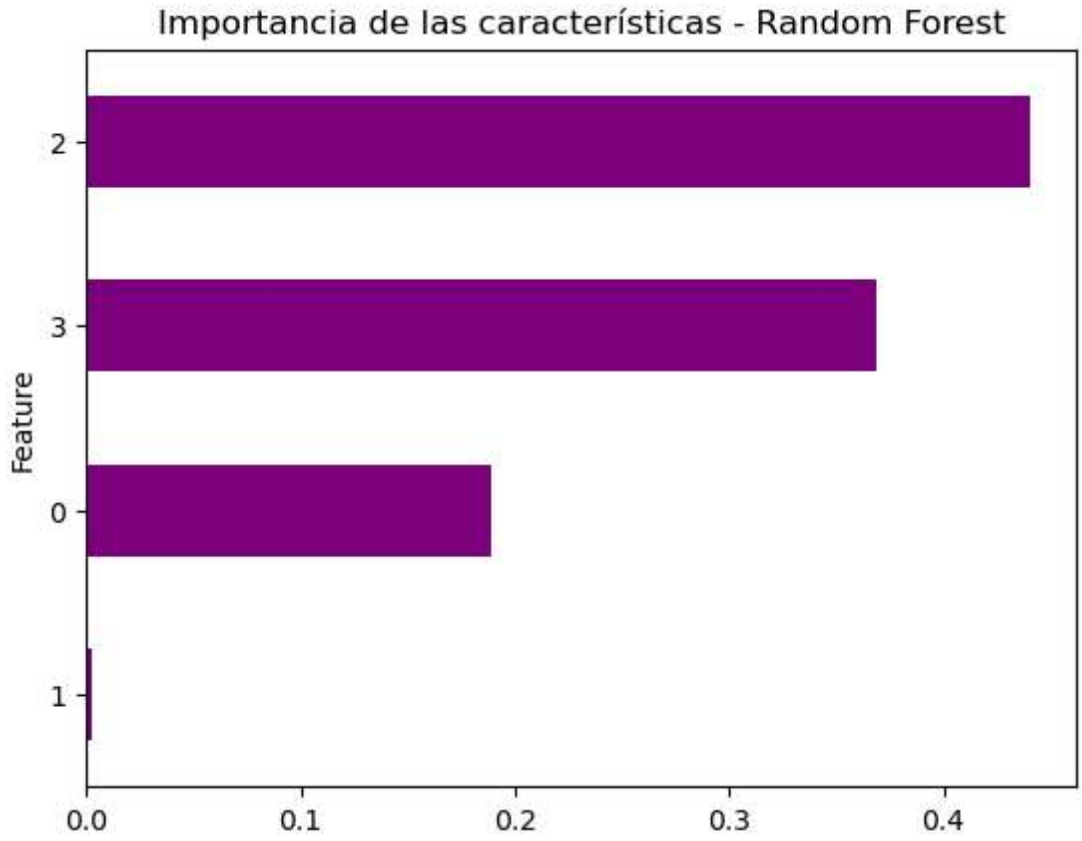
4. Evaluación

Ahora utilizamos las variables objetivo del conjunto de test para comparar con la predicción que hemos hecho anteriormente.

```python
# evaluación
print('\n---Resultados del modelo (Random Forest)---')
print(f'Accuracy: {accuracy_score(y_test_rf, y_pred_rf)}')
print(classification_report(y_test_rf, y_pred_rf))

# visualización
cm = confusion_matrix(y_test_rf, y_pred_rf, labels=model_rf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model_rf.classes_)
disp.plot(cmap=plt.cm.Purples)
plt.title('Matriz de confusión - Random Forest')
plt.show()

importances = model_rf.feature_importances_
feature_names = df.columns.drop(4)
feature_df = pd.DataFrame({"Feature": feature_names, "Importance": importances})
feature_df = feature_df.sort_values(by="Importance", ascending=True)
feature_df.plot(kind="barh", x="Feature", y="Importance", color='purple', legend=False)
plt.title("Importancia de las características - Random Forest")
plt.show()
```

```
---Resultados del modelo (Random Forest)---
Accuracy: 0.9333333333333333
              precision    recall  f1-score   support

     Benigno       0.90      0.90      0.90        10
     Maligno       0.90      0.90      0.90        10
      Normal       1.00      1.00      1.00        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```





También hemos creado una matriz de confusión y gráfico de barras que muestra el peso de las variables al hacer la predicción.