TC3006C.101

Análisis y Reporte sobre el desempeño del modelo

Módulo 2, Portafolio Análisis

Luis Cano Irigoyen – A00827178
9-8-2022

Implementación seleccionada:

Para en análisis sobre el desempeño de un modelo en un set de daos, se utilizará la implementación de la actividad 2 del portafolio: *Uso de framework o biblioteca de aprendizaje máquina para la implementación de una solución*.

Modelo:

```
DecisionTreeClassifier(random_state=0)
```

Base de Datos:

```
"https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
```

Separación y evaluación del modelo con un conjunto de prueba y un conjunto de validación

Inicialmente los datos se separan en conjuntos de entrenamiento y prueba

```
x_train, x_test, y_train, y_test = train_test_split(input, output, test_size
= 0.2, random_state = 42)
```

El modelo se entrena con los datos de entrenamiento

```
model.fit(x_train, y_train)
```

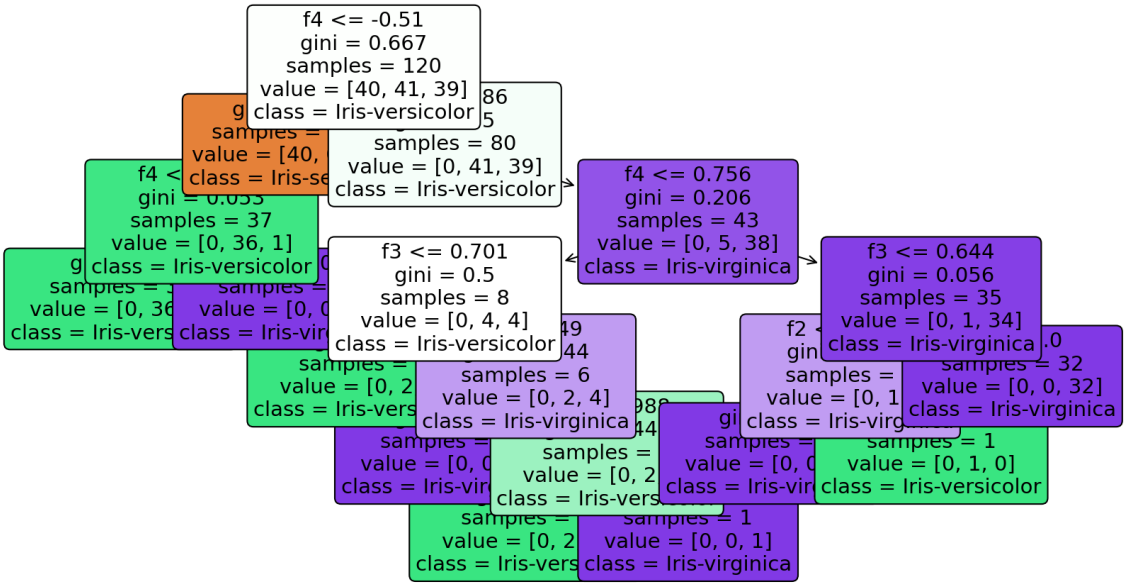
Se realizan predicciones con los datos de prueba

```
y_pred = model.predict(x_test)
```

Y se calculan las precisiones del modelo

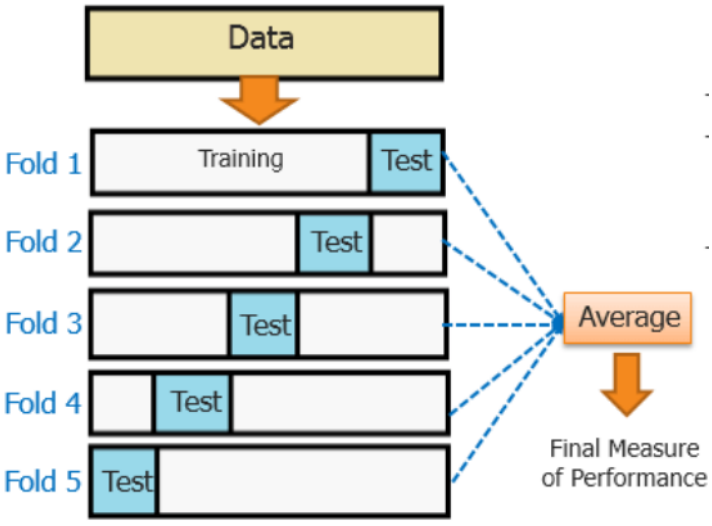
```
Accuracy of the models with the subsets
             Nombre  Train Accuracy  Test Accuracy
0  Decision Tree b             1.0            1.0
```

El árbol de decisión resultado:



Parece ser un modelo con precisión perfecta. Sin embargo, estos cálculos son susceptibles a ser incorrectos por “suerte”. Un modelo podría quedar con precisión muy alta o baja con cierta división en subconjuntos de entrenamiento y prueba, pero que no la vuelva a alcanzar si los subconjuntos de datos se partieran de diferente manera. Así uno podría terminar con un modelo *overfitted* sin darse cuenta.

Por ello realizamos una validación cruzada, entrenando al modelo en cada uno de los subconjuntos, y calculando la precisión final cómo el promedio de cada una de las precisiones obtenidas.



En este caso se utilizó un *KFold*, realizando 5 divisiones de los datos. Este se utiliza para realizar la *cross validation* de la siguiente manera:

```
cv_results = cross_validate(model, input, output, cv=kfold, scoring=scorer)
```

Y se obtuvieron los siguientes resultados:

```
Accuracy of the Decision Tree b model with k-fold cross validation
K-fold test accuracies:
 [1.         0.96666667 0.93333333 0.93333333 0.93333333]
K-fold train accuracies:
 [1. 1. 1. 1. 1.]
Mean test accuracy: 0.9533333333333335
Mean train accuracy: 1.0
Standard deviation:  0.02666666666666666
Variance:  0.0007111111111111108
Bias:  0.04666666666666652
```

Entonces vemos que en este modelo la precisión no era perfecta, dicha solo ocurre en el primer *fold*. La precisión termina siendo el promedio de todas, igual a 95.3%.

Diagnóstico y explicación el grado de bias o sesgo

Para calcular el *Bias* de los datos de prueba realizamos:

$$Bias = 1 - test\_accuracy$$

Aprovechando el *cross validation* que realizamos, podemos calcular el *Bias* utilizando la precisión promedio de los 5 *folds*:

```
1 - cv_results['test_score'].mean()
```

Y obtenemos

```
Bias:  0.04666666666666652
```

Este es un Bias bajo, indicando que hay una buena precisión en las clasificaciones del modelo. Sin embargo, no es perfecto y se podría mejorar.

Diagnóstico y explicación el grado de varianza

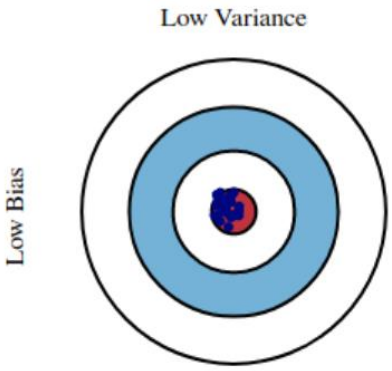
Para el cálculo de la Varianza de igual manera podemos utilizar el *cross validation* realizado, y obtener la Varianza después de los 5 *KFolds* con la siguiente fórmula:

```
cv_results['test_score'].var()
```

Y con esto obtenemos

```
Variance:  0.0007111111111111108
```

Esta es una Varianza muy baja, lo cual indica que hay una alta consistencia entre las predicciones de los modelos. Este valor es bueno.



Diagnóstico y explicación el nivel de ajuste del modelo

Para determinar el nivel de ajuste utilizamos el *cross validation* del modelo. Algo que es útil es considerar la precisión de los datos de entrenamiento para compararla con los de prueba:

```
K-fold test accuracies:
 [1.         0.96666667 0.93333333 0.93333333 0.93333333]
K-fold train accuracies:
 [1. 1. 1. 1. 1.]
Mean test accuracy: 0.9533333333333335
Mean train accuracy: 1.0
```

Como debería ser, la precisión de prueba suele encontrarse por debajo de la precisión de entrenamiento.

En promedio, las precisiones son de 1.0 para prueba y 0.95 para entrenamiento. Esta es una diferencia muy pequeña entre las precisiones por lo cual no se sugiere que haya *overfit*.

De igual manera, ambas son precisiones altas, el modelo suele estar acertado en sus clasificaciones, por lo cual tampoco se considera que esté *underfit*.

En conclusión, podríamos determinar que el modelo está *fit*.

Técnicas de regularización o ajuste de parámetros para mejorar el desempeño del modelo

En la regularización y ajuste de parámetros lo que buscamos es penalizar el tamaño y complejidad del modelo, para buscar mejorar su desempeño y obtener mejores resultados.

En general, en un árbol de decisión la complejidad esta controlada por la *max_depth*, la máxima profundidad del árbol. El valor por default es *None*, en donde los nodos continuan expandiéndose hasta que todas las hojas son puras o que todas contengan menos de *min_samples_split* muestras.

El modificar la *max_depth* nos permite compensar entre un modelo más *overfitted* o *underfitted*. Aumentar la profundidad hace al modelo más expresivo, pero se arriesga a tener un modelo *overfitted*.

Otros parámetros que vale la pena considerar modificar son *min_samples_leaf* y *min_samples_split*, los cuales nos permiten aplicar restricciones al nivel de las hojas o nodos.

También podemos cambiar la fórmula matemática de criterio del modelo, la cual determina dónde un árbol hace ramificación. Para este parámetro se puede seleccionar entre *Gini* y *Entropy*.

Son múltiples parámetros que podemos ajustar, a variedad de valores, por lo cual modificarlos “al tanteo” no resulta una buena decisión y optamos por realizar un *Grid Search*. Entonces se define un *grid* con todas las opciones de parámetros y valores que uno quiere analizar el modelo:

```
param_grid = {
    'max_depth': [3, 4, 5, 6, 7],
    'min_samples_leaf': [1, 2, 3, 4],
    'min_samples_split': [2, 3, 4, 5, 6],
    'criterion': ['gini', 'entropy'],
}
```

Con este se corre un *GridSearchCV* o *RandomizedSearchCV* para determinar con *cross validation* cuales son los mejores parámetros para el modelo, y finalmente se obtienen en un output.

Entre los mejores modelos que encontré están los siguientes dos

MODELO FINAL 1

Hiperparámetros encontrados:

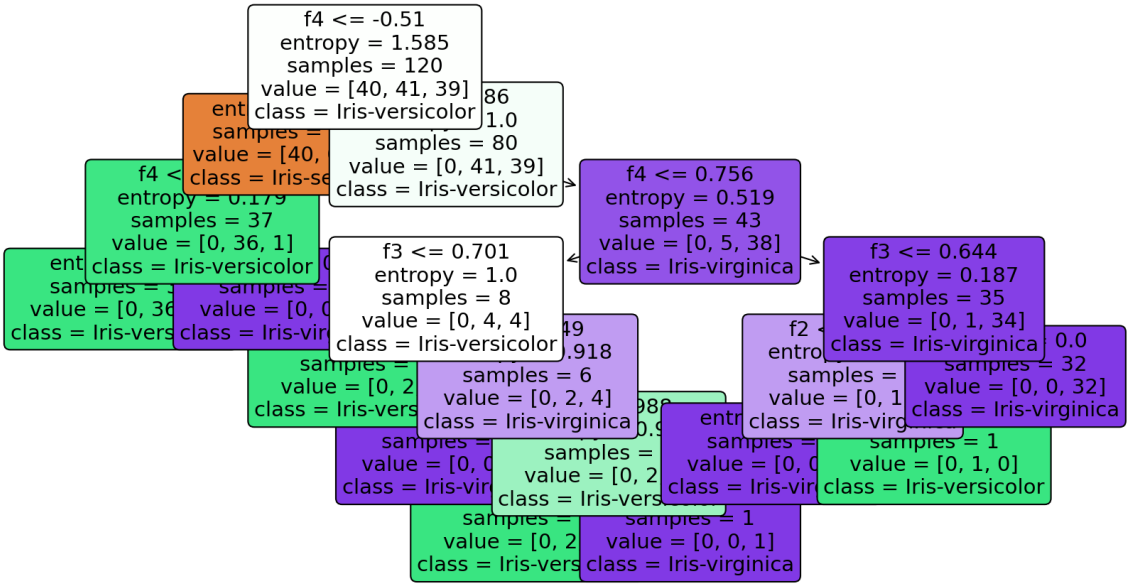
```
Best parameters found by grid search:
{'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 6, 'criterion': 'entropy'}
```

Definición del modelo:

```
DecisionTreeClassifier(criterion = 'entropy', max_depth = 6,
min_samples_leaf = 1, min_samples_split = 2, random_state = 0)
```

Resultados:

```
Accuracy of the Decision Tree Final 1 model with k-fold cross validation
K-fold test accuracies:
 [1.         1.         0.93333333 0.93333333 0.93333333]
K-fold train accuracies:
 [1. 1. 1. 1. 1.]
Mean test accuracy: 0.9600000000000002
Mean train accuracy: 1.0
Standard deviation:  0.03265986323710904
Variance:  0.0010666666666666665
Bias:  0.039999999999999813
```



MODELO FINAL 2

Hiperparámetros encontrados:

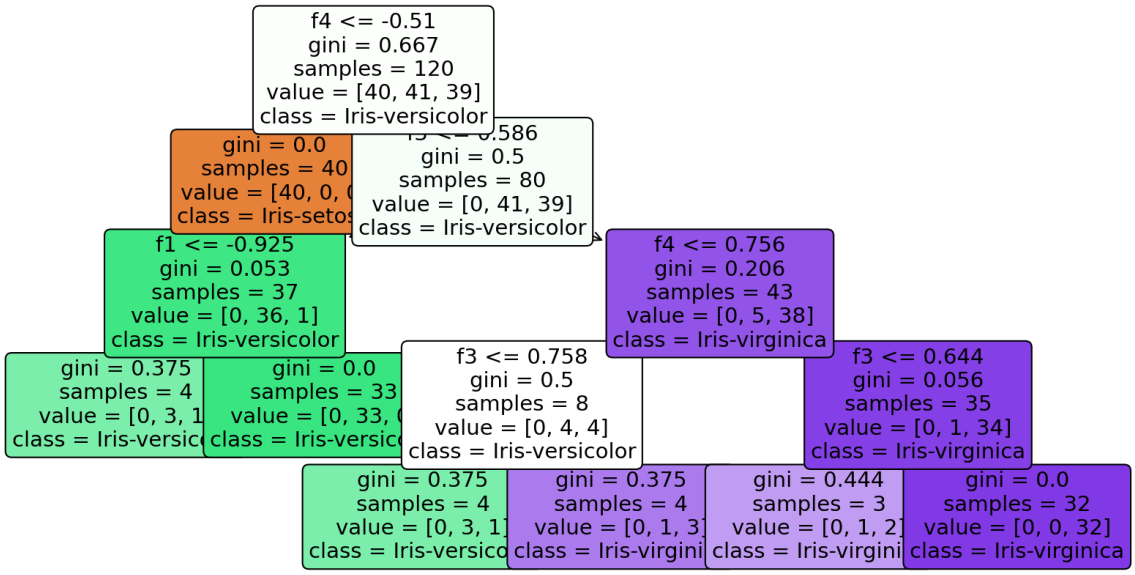
```
Best parameters found by the grid search:
{'min_samples_split': 6, 'min_samples_leaf': 3, 'max_depth': 4, 'criterion': 'gini'}
```

Definición del modelo:

```
DecisionTreeClassifier(criterion = 'gini', max_depth = 4, min_samples_leaf =
3, min_samples_split = 6, random_state = 0)
```

Resultados:

```
Accuracy of the Decision Tree Final 2 model with k-fold cross validation
K-fold test accuracies:
 [1.         1.         0.93333333 0.96666667 0.96666667]
K-fold train accuracies:
 [0.96666667 0.96666667 0.99166667 0.975      0.975     ]
Mean test accuracy: 0.9733333333333334
Mean train accuracy: 0.975
Standard deviation:  0.024944382578492935
Variance:  0.0006222222222222219
Bias:  0.026666666666666616
```



Conclusiones del resultado

El Modelo Final 1, *MF1*, resulta bastante similar al modelo original de la implementación (el DecisionTreeClassifier sin parámetros), pero con ligeras mejoras. Ambos cuentan con 19 nodos en total, ambos terminaron con una profundidad de 6 niveles de decisiones, mismos parámetros en *min_samples_leaf* y *min_samples_split* (considerando los parámetros por default que utiliza el modelo original), y sólo cambiando el criterio de decisión entre *Gini* y *Entropy*.

El usar *Entropy* le permitió al *MF1* alcanzar la misma precisión promedio con los datos de entrenamiento con una precisión de 0.0066 (0.6%) más alta en los datos de prueba. De igual manera hubo cambios mínimos en el *Bias* y *Varianza* del *MF1*, en dónde el *Bias* fue menor al original y la *Varianza* aumentó muy poco.

Por el otro lado el Modelo Final 2, *MF2*, si cuenta con más cambios del original. Cuenta con 2 niveles menos, una menor cantidad de nodos, parámetros de *min_samples_leaf* y *min_samples_split* diferentes, y solo mantiene igual el criterio de decisión de *Gini*.

Con estos cambios se pierde precisión en los datos de entrenamiento, terminando con un 0.975, pero se alcanza la precisión más alta en los datos de prueba con 0.973, aumentando por 2% a comparación del modelo original. Adicionalmente ambos valores de *Bias* y *Varianza* resultan los menores de los tres modelos.

Considero que el *MF2* es el que alcanzó una mayor mejora de desempeño ya que es el que cuenta con un mejor nivel de ajuste y mayor precisión en los datos de prueba; el *MF1* de igual manera tiene buen nivel de ajuste, pero es ligeramente más *overfitted* que el *MF2*. También, al tener menos niveles de profundidad y menos nodos este modelo resulta el de menor tamaño y menos complejo. Asimismo, los bajos valores de *Bias* y *Varianza* del *MF2* nos muestran como de los tres, este es el que tiene mejor precisión en las clasificaciones y mayor consistencia en las predicciones entre los modelos.