


▼ Importar librerías

```
1 import pandas as pd
2 import numpy as np
3 from sklearn import metrics
4 import matplotlib.pyplot as plt
```

▼ Dataset

```
1 from sklearn import datasets
2 iris = datasets.load_iris()
3 df = pd.DataFrame(iris.data, columns = iris.feature_names)
4 df['target'] = iris.target
5 X = iris.data
6 df.sample(4)
```



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
22	4.6	3.6	1.0	0.2	0
4	5.0	3.6	1.4	0.2	0
143	6.8	3.2	5.9	2.3	2
37	4.9	3.6	1.4	0.1	0

```
1 df.isna().sum()
```

```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
dtype: int64
```

```
1 df.shape
```

```
(150, 5)
```

```
1 df['target'].unique()
array([0, 1, 2])
```

▼ Conjunto de (Train/Test/Validation)

Se dividen los datos en sets de entrenamiento, validación y prueba con un 80%, 10% y 10% respectivamente de toda la base de datos, esto con el objetivo de tener la mayor cantidad de datos para entrenamiento, evitando underfitting, sin dejar muy pocos para la prueba y validación de rendimiento del modelo.

```
1 from sklearn.model_selection import train_test_split
2
3 X = df.drop(['target'], axis = 1)
4 y = df['target']
5
6 train_ratio = 0.80
7 test_ratio = 0.10
8 validation_ratio = 0.10
9
10 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_ratio)
11
12 X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=validation_ratio/(train_ratio+test_ratio))
13
14 print("X_train : ",X_train.shape)
15
16 print("X_test : ",X_test.shape)
17
18 print("X_valid : ",X_valid.shape)

X_train : (119, 4)
X_test : (15, 4)
X_valid : (16, 4)
```

▼ Implementación del Modelo

El modelo a implementar fue un Decision Tree mediante la librería Scikit Learn. Primero se generó el modelo con los datos de entranamiento sin considerar alguna alteración para mejorar el rendiemiento del mismo.

```
1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn import tree
3 classifier = tree.DecisionTreeClassifier(criterion='gini')
4
5 classifier = classifier.fit(X_train, y_train)
```

▼ Grado de bias o sesgo y Ajuste del modelo

Para determinar el ajuste del modelo nos fijaremos en la accuracy del modelo, puesto que al probar el modelo entrenado con los datos de entranamiento si el resultado de la accuracy es bajo, caemos en underfitting; asimismo, si al probar el modelo con datos de prueba el accuracy es bajo con un alto accuracy en con los datos de entranamiento, caemos en un overfitting.

```
1 # Prueba del modelo con los datos de entranamiento:
2 test_pred_train = classifier.predict(X_train)
3
4
5 print("Accuracy:" )
6
7 metrics.accuracy_score(y_train, test_pred_train)
```

```
Accuracy:
1.0
```

Tener un accuracy alto en esta instancia nos indica que no caimos en un underfitting; sin embargo, aun el modelo es susceptible a tener un overfitting.

```
1 # Prueba del modelo con los datos de prueba:
2 test_pred_valid = classifier.predict(X_valid)
3
4 print("Accuracy: " )
5
6 metrics.accuracy_score(y_valid, test_pred_valid)
```

```
Accuracy:  
0.9375
```

Una accuracy de 87.5% es un resultado favorable, lo que nos indica que el modelo actual no cae en underfitting u overfitting.

```
1 1-metrics.accuracy_score(y_valid, test_pred_valid)  
  
0.0625
```

Para medir el sesgo se puede observar el complemento de la accuracy que está relacionado con los valores no acertados del modelo. La accuracy se puede interpretar como la ausencia del bias; por lo tanto, el complemento nos indica el error que hubo, es decir, el bias que está afectando las predicciones del modelo. En este caso un bias de 0.125 es bajo.

▼ Grado de varianza


Para medir el grado de varianza se utilizan dos parámetros importantes del modelo: la profundidad del árbol de decisión y el tamaño de las muestras.

```
1 from sklearn.model_selection import cross_val_score  
2 import numpy as np  
3 import matplotlib.pyplot as plt  
4  
5 # function for fitting trees of various depths on the training data using cross-validation  
6 def run_cross_validation_on_trees(X, y, tree_depths, cv=5, scoring='accuracy'):  
7     cv_scores_list = []  
8     cv_scores_std = []  
9     cv_scores_mean = []  
10    accuracy_scores = []  
11    for depth in tree_depths:  
12        tree_model = DecisionTreeClassifier(max_depth=depth)  
13        cv_scores = cross_val_score(tree_model, X, y, cv=cv, scoring=scoring)  
14        cv_scores_list.append(cv_scores)  
15        cv_scores_mean.append(cv_scores.mean())  
16        cv_scores_std.append(cv_scores.std())  
17        accuracy_scores.append(tree_model.fit(X, y).score(X, y))  
18    cv_scores_mean = np.array(cv_scores_mean)
```

```
19     cv_scores_std = np.array(cv_scores_std)
20     accuracy_scores = np.array(accuracy_scores)
21     return cv_scores_mean, cv_scores_std, accuracy_scores
22
23 # function for plotting cross-validation results
24 def plot_cross_validation_on_trees(depths, cv_scores_mean, cv_scores_std, accuracy_scores, title):
25     fig, ax = plt.subplots(1,1, figsize=(15,5))
26     ax.plot(depths, cv_scores_mean, '-o', label='mean cross-validation accuracy', alpha=0.9)
27     ax.fill_between(depths, cv_scores_mean-2*cv_scores_std, cv_scores_mean+2*cv_scores_std, alpha=0.2)
28     ylim = plt.ylim()
29     ax.plot(depths, accuracy_scores, '-*', label='train accuracy', alpha=0.9)
30     ax.set_title(title, fontsize=16)
31     ax.set_xlabel('Tree depth', fontsize=14)
32     ax.set_ylabel('Accuracy', fontsize=14)
33     ax.set_ylim(ylim)
34     ax.set_xticks(depths)
35     ax.legend()
36
37 # fitting trees of depth 1 to 24
38 sm_tree_depths = range(1,25)
39 sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores = run_cross_validation_on_trees(X_train, y_train, sm_tree_depths)
40
41 # plotting accuracy
42 plot_cross_validation_on_trees(sm_tree_depths, sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores,
43                               'Accuracy per decision tree depth on training data')
```

Accuracy per decision tree depth on training data

La gráfica nos indica el rendimiento del decision tree con muchas profundidades, esto nos ayuda a decidir la profundidad del modelo para alcanzar un resultado óptimo sin desperdiciar costo computacional. Como se puede observar, después de una profundidad de 4 el modelo no tiene mucha variabilidad en cuanto su rendimiento tanto para los datos de entrenamiento como los de prueba.



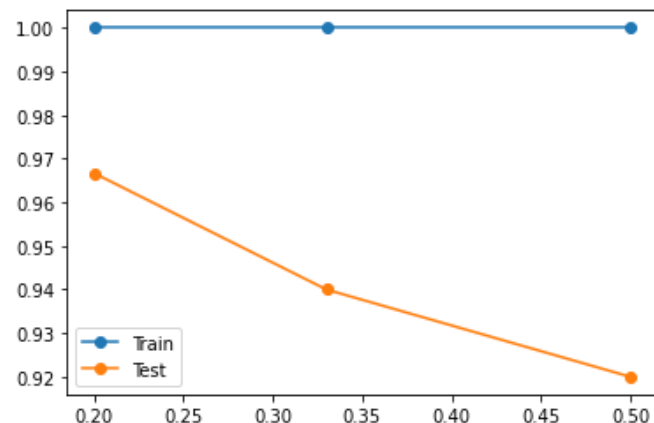
```

1 # evaluate decision tree performance on train and test sets with different tree depths
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import accuracy_score
4 from sklearn.tree import DecisionTreeClassifier
5 from matplotlib import pyplot
6
7 sizes = [0.2, 0.33, 0.5]
8
9
10 # define lists to collect scores
11 train_scores, test_scores = list(), list()
12 # define the tree depths to evaluate
13 values = [i for i in range(len(sizes))]
14 # evaluate a decision tree for each depth
15 for i in range(len(values)):
16     # configure the model
17
18     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=sizes[i])
19
20     model = DecisionTreeClassifier(criterion = 'gini')
21     # fit model on the training dataset
22     model.fit(X_train, y_train)
23     # evaluate on the train dataset
24     train_yhat = model.predict(X_train)
25     train_acc = accuracy_score(y_train, train_yhat)
26
27     train_scores.append(train_acc)
28     # evaluate on the test dataset
29     test_yhat = model.predict(X_test)
30     test_acc = accuracy_score(y_test, test_yhat)
31     test_scores.append(test_acc)
32     # summarize progress
33     print('>%d, train: %.3f, test: %.3f' % (i, train_acc, test_acc))
34 # plot of train and test scores vs tree depth
35 pyplot.plot(sizes, train_scores, '-o', label='Train')
36 pyplot.plot(sizes, test_scores, '-o', label='Test')

```

```
37 pyplot.legend()
38 pyplot.show()
```

```
>0, train: 1.000, test: 0.967
>1, train: 1.000, test: 0.940
>2, train: 1.000, test: 0.920
```



La gráfica nos muestra el rendimiento del modelo considerando diferentes tamaños de muestras. Vemos que el rendimiento del modelo al probar con el conjunto de prueba decrece a medida que hacemos el conjunto de entrenamiento más pequeño. La variabilidad de este suceso se explica por utilizar cada vez menos datos de entrenamiento para construir el modelo ya que esto perjudica directamente las predicciones del modelo.

▼ Implementación del modelo final

Finalmente se implementa el modelo tomando en consideración los parámetros para incrementar el rendimiento del modelo:

- profundidad mayor o igual a 4
- tamaño del conjunto de prueba 33% de los datos

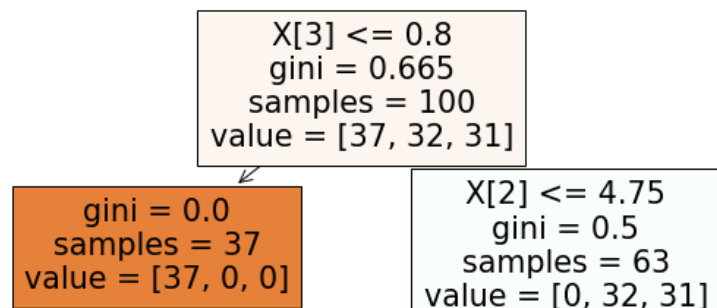
```
1 from sklearn.model_selection import train_test_split
2
3 X = df.drop(['target'], axis = 1)
4 y = df['target']
5
6 train_ratio = 0.67
7 test_ratio = 0.33
```

```
8
9 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_ratio)
10
11 print("X_train : ",X_train.shape)
12
13 print("X_test : ",X_test.shape)
```

```
    X_train : (100, 4)
    X_test  : (50, 4)
```

```
1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn.metrics import classification_report
3 from sklearn import tree
4
5 clf = DecisionTreeClassifier(max_depth = 4,random_state=0)
6 clf.fit(X_train, y_train)
7 y_pred=clf.predict(X_test)
```

```
1 fig = plt.figure(figsize=(15,10))
2 _ = tree.plot_tree(clf,
3                     filled=True)
```

▼ Métricas de Rendimiento

value = [0, 29, 0]

value = [0, 3, 31]

Se utilizaron métricas como accuracy, recall, f1-score y support para determinar el rendimiento del modelo.

| gini = 0.5 |

gini = 0.0

```

1 print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
2
3 print(classification_report(y_test, y_pred))

```

```

Accuracy: 0.94

```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	0.89	0.94	0.92	18
2	0.94	0.89	0.92	19
accuracy			0.94	50
macro avg	0.95	0.95	0.95	50
weighted avg	0.94	0.94	0.94	50

```

1 print("Accuracy:", metrics.accuracy_score(y_test, y_pred))

```

```

Accuracy: 0.94

```

El modelo cumple con estandares aceptables para ser un buen modelo.

- Mantiene una accuracy de 94%
- Recall del 100%
- F1-Score del 100%

Al considerar los nuevos parámetros óptimos del modelo se puede notar aumento del 0.025%. Esta mejora apesar de no ser realmente significativa se puede considerar como un aumento decente para modelos más complejos o inclusive uan diferencia favorable al utilizar el modelo para predecir grandes bases de datos.

```
1 y_test=y_test.to_list()

1 for i in range(len(y_test)):
2     print('Predicción: ', y_pred[i], ' | Valor Esperado: ', y_test[i])
```

```
Predicción: 1 | Valor Esperado: 1
Predicción: 2 | Valor Esperado: 2
Predicción: 2 | Valor Esperado: 2
Predicción: 0 | Valor Esperado: 0
Predicción: 2 | Valor Esperado: 2
Predicción: 2 | Valor Esperado: 2
Predicción: 2 | Valor Esperado: 2
Predicción: 2 | Valor Esperado: 2
Predicción: 1 | Valor Esperado: 1
Predicción: 1 | Valor Esperado: 1
Predicción: 2 | Valor Esperado: 2
Predicción: 2 | Valor Esperado: 2
Predicción: 0 | Valor Esperado: 0
Predicción: 1 | Valor Esperado: 1
Predicción: 1 | Valor Esperado: 1
Predicción: 0 | Valor Esperado: 0
Predicción: 0 | Valor Esperado: 0
Predicción: 1 | Valor Esperado: 1
Predicción: 2 | Valor Esperado: 2
Predicción: 2 | Valor Esperado: 2
Predicción: 1 | Valor Esperado: 1
Predicción: 1 | Valor Esperado: 1
Predicción: 0 | Valor Esperado: 0
Predicción: 2 | Valor Esperado: 2
Predicción: 0 | Valor Esperado: 0
Predicción: 2 | Valor Esperado: 2
Predicción: 0 | Valor Esperado: 0
Predicción: 0 | Valor Esperado: 0
Predicción: 2 | Valor Esperado: 1
Predicción: 2 | Valor Esperado: 2
Predicción: 2 | Valor Esperado: 2
Predicción: 1 | Valor Esperado: 1
Predicción: 0 | Valor Esperado: 0
Predicción: 2 | Valor Esperado: 2
Predicción: 2 | Valor Esperado: 2
Predicción: 2 | Valor Esperado: 2
```

Predicción:	0		Valor Esperado:	0
Predicción:	2		Valor Esperado:	2
Predicción:	1		Valor Esperado:	1
Predicción:	1		Valor Esperado:	1
Predicción:	0		Valor Esperado:	0
Predicción:	1		Valor Esperado:	1
Predicción:	0		Valor Esperado:	0
Predicción:	1		Valor Esperado:	1
Predicción:	0		Valor Esperado:	0
Predicción:	1		Valor Esperado:	1
Predicción:	1		Valor Esperado:	1
Predicción:	1		Valor Esperado:	1
Predicción:	1		Valor Esperado:	1
Predicción:	1		Valor Esperado:	2
Predicción:	1		Valor Esperado:	1
Predicción:	1		Valor Esperado:	2

[Colab paid products](#) - [Cancel contracts here](#)

