Clasificación: Decision Tree

Minería de Datos
Zavala Roman Irvin Eduardo 1270771

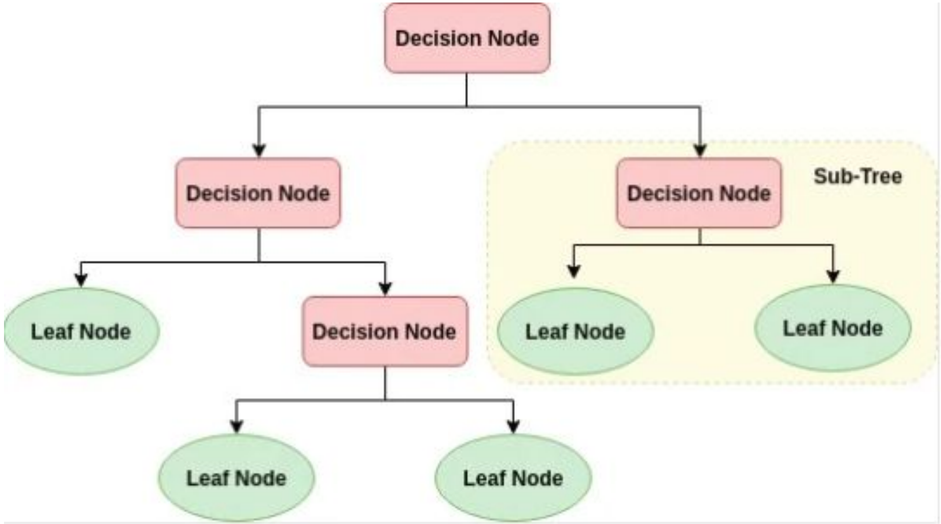
Dataset: Calidad de vinos

```
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   object
dtypes: float64(11), object(1)
memory usage: 150.0+ KB
```

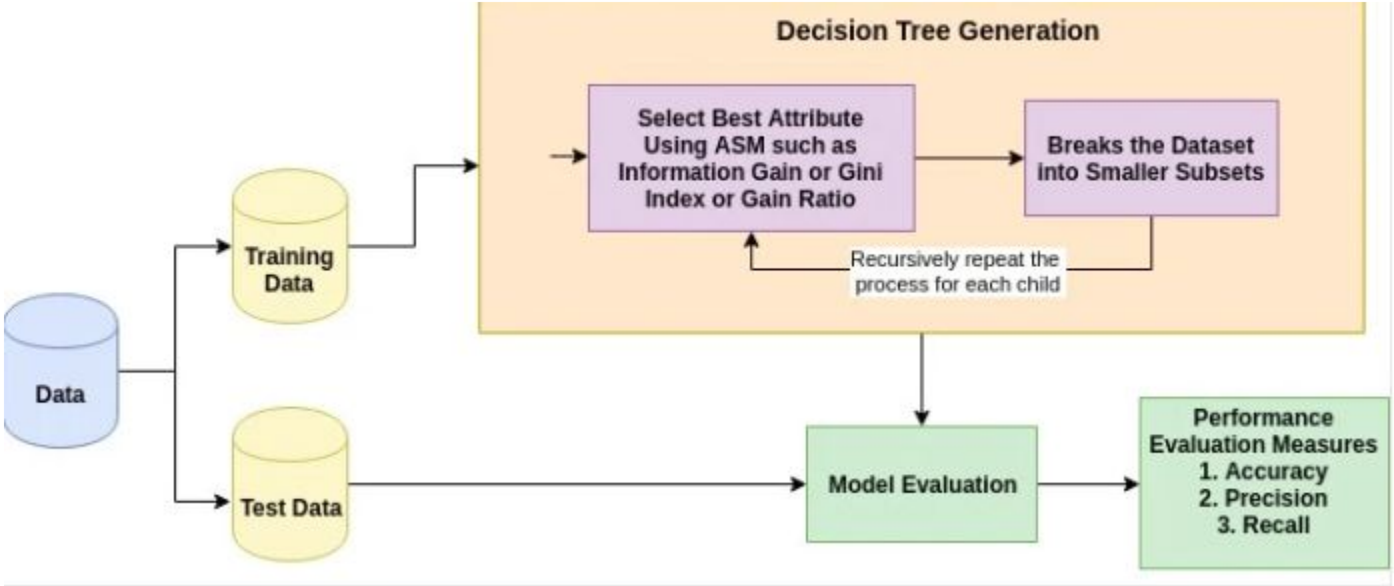
```
   fixed acidity  volatile acidity  citric acid  ...  sulphates  alcohol  quality
0            7.4              0.70         0.00  ...       0.56      9.4      bad
1            7.8              0.88         0.00  ...       0.68      9.8      bad
2            7.8              0.76         0.04  ...       0.65      9.8      bad
3           11.2              0.28         0.56  ...       0.58      9.8     good
4            7.4              0.70         0.00  ...       0.56      9.4      bad
```

Clasificador por árboles de decisión

Los árboles tienen una gran cantidad de aplicaciones en ML, una es clasificación. Su característica fundamental es que son algoritmos de caja blanca, por lo que es totalmente explicable.



Idea del algoritmo



*ASM: Attribute Selection Measure

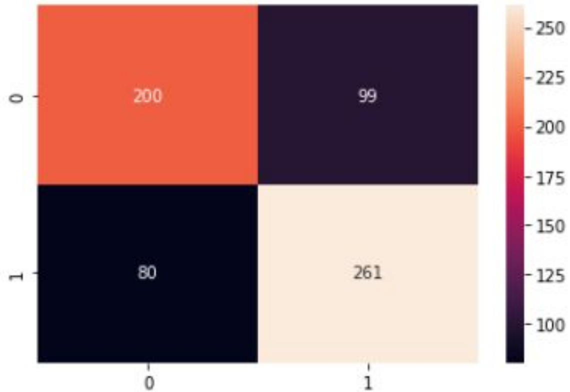
sklearn.tree.DecisionTreeClassifier

class `sklearn.tree.DecisionTreeClassifier`(**, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0*) [source]

- criterion*{"gini", "entropy", "log_loss"}, default="gini"*
- splitter*{"best", "random"}, default="best"*
- max_depth: *int, default=None*
- min_samples_split: *int or float, default=2*
- min_samples_leaf: *int or float, default=1*
- min_weight_fraction_leaf: *float, default=0.0*
- max_features: *int, float or {"auto", "sqrt", "log2"}, default=None*
- random_state: *int, RandomState instance or None, default=None*
- max_leaf_nodes: *int, default=None*
- min_impurity_decrease: *float, default=0.0*
- class_weight: *dict, list of dict or "balanced", default=None*
- ccp_alpha: *non-negative float, default=0.0*

Ejemplo de uso sin optimizar

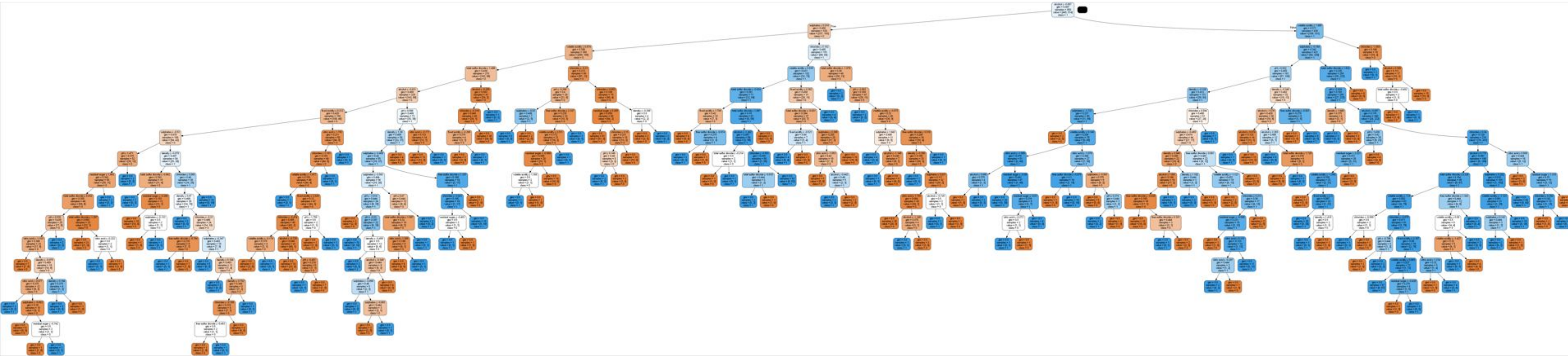
```
dataset = pd.read_csv("wine.csv", on_bad_lines='skip')
labels = dataset.columns
X = dataset.loc[:, labels[:-1]].values
y = dataset.loc[:, labels[-1]].values

sc = StandardScaler()
X_train,X_test,y_train,y_test = train_test_split(X, y, train_size=0.6)
model = DecisionTreeClassifier()
model = model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:",accuracy_score(y_test,y_pred))
cm = confusion_matrix(y_test, y_pred)
fig = plt.figure()
ax = sn.heatmap(cm,annot=True, fmt='g')
plt.show()
```




```
Accuracy: 0.7203125
```

Visualización del árbol

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
dot_data = StringIO()
export_graphviz(model , out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = labels[:-1],class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('arbol_vinitos.png')
Image(graph.create_png())
```

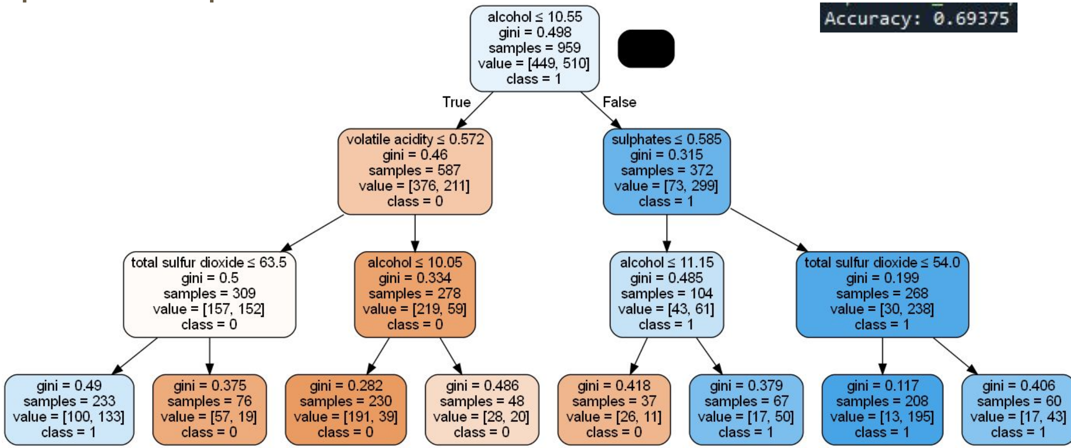


Argumentos importantes

- **criterion:** Puede variar por dataset pero cambiar entre gini y entropy puede dar resultados distintos
- **max_depth**: Profundidad del árbol, si se limita se pierde accuracy a cambio de un árbol mejor visualizable
- **max_features**: Cantidad de atributos a considerar al realizar una decisión
- **splitter**: Estrategia para escoger la partición por nodo

Information gain: Entropia por categoría para obtener el IG por feature

Gini index: Combina el ruido por categoría para obtener el ruido por feature



Accuracy: 0.69375

Ventajas

- Fáciles de visualizar
- Poco preprocesamiento
- Es no paramétrico
- Caja blanca

Desventajas

- Sensible al ruido
- Pequeñas variaciones generan resultados distintos
- Son sesgados por el dataset

Clase creada para clasificar

```
class classifier():
    """
        Parameters
        ----------
        X: array of data
        y : array of classes
        split_method: str ['holdout', 'random_subsampling', 'kfold', 'leaveoneout','stratifiedkfold']
        classifier: str ['logistic', 'kneighbors', 'decisiontree', 'gaussianNB', 'sgd', 'adaboost',  'mlp']

        k {OPTIONAL}: Obligatory when use [kfold, random_subsampling, stratifiedkfold]
        train_size {OPTIONAL}: Obligatory when use holdout

        Methods
        -------

        classify(self): Classifies and get metrics

        getMetrics(self): Return metrics outside class
    """
```

Idea de la clase

```
#Por cada split se crea un cv
cv = KFold(n_splits=self.k)

#Por cada clasificador se crea un modelo
model = LogisticRegression()

#Si el cv es iterable
for train index , test index in cv.split(self.X, y):
    X train , X test = X scaled[train index,:],X scaled[test index,:]
    y train , y test = self.y[train index] , self.y[test_index]
    y pred = model.fit(X_train,y_train).predict(X_test)
    self.y test = y test
    self.y pred = y pred
    cm,accuracy,error rate,sensitivity,specifity,precision = self.getInternalMetrics()
    metrics['accuracy'] = np.append(metrics['accuracy'], accuracy)
    metrics['error rate'] = np.append(metrics['error rate'], error rate)
    metrics['sensitivity'] = np.append(metrics['sensitivity'], sensitivity)
    metrics['specifity'] = np.append(metrics['specifity'], specifity)
    metrics['precision'] = np.append(metrics['precision'], precision)
self.metrics[0] = metrics['accuracy'].mean()
self.metrics[1] = metrics['error rate'].mean()
self.metrics[2] = metrics['sensitivity'].mean()
self.metrics[3] = metrics['specifity'].mean()
self.metrics[4] = metrics['precision'].mean()
```

```python
def tableMethod(method):
    from tabulate import tabulate
    table = []
    head_table = ["SPLIT METHOD","accuracy","error rate", "sensitivity", "specifity","precision"]
    split_methods = [ 'holdout', 'holdout','random_subsampling' , 'kfold', 'leaveoneout' , 'stratifiedkfold' ]
    table.append(head_table)
    k = 0
    train_size = 0
    max_prom = 0
    best_row = [0,99,0,0,0]
    best_split =''
    for i in range(len(split_methods)):
        if(i == 0):
            train_size = 0.6
            k = 0
        elif(i == 1):
            train_size = 1
            k = 0
        elif(i == 2):
            train_size = 0.6
            k = 30
        else:
            train_size = 0
            k = 10
        c = classifier(X, y, split_methods[i], method, k, train_size)
        c.classify()
        accuracy,error_rate,sensitivity,specifity,precision = c.getMetrics()
        row = [split_methods[i], accuracy,error_rate,sensitivity,specifity,precision]
        if(row[1] > best_row[0] and row[2] < best_row[1] and
           row[3] > best_row[2] and row[4] > best_row[3] and row[5] > best_row[4]):
            best_row = row[1:]
            best_split = split_methods[i] + '| k = ' + str(k) + '| train_size = ' + str(train_size)
        table.append(row)
    print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
    return best_row, best_split
```

Buscando la mejor combinación

```
dataset = pd.read_csv("wine.csv", on_bad_lines='skip')
labels = dataset.columns
X = dataset.loc[:, labels[:-1]].values
y = dataset.loc[:, labels[-1]].values
import timeit
best_row = [0,99,0,0,0]
best_combination= ''
split_methods = ['holdout', 'random subsampling', 'kfold', 'stratifiedkfold',]
classifiers = ['logistic', 'kneighbors', 'decisiontree', 'gaussianNB', 'sgd', 'adaboost',  'mlp']
for i in range(len(classifiers)-2):
    print("________________{}________________".format(classifiers[i]))
    start = timeit.default_timer()
    aux, split = tableMethod(classifiers[i])
    end = timeit.default_timer()
    if(aux[0] > best_row[0] and aux[1] < best_row[1] and
       aux[2] > best_row[2] and aux[3] > best_row[3] and
       aux[4] > best_row[4] and split != 'holdout| k = 0| train_size = 1'):
        best_row = aux
        best_combination = classifiers[i] + '|' + split
    print("Tiempo de ejecucion:", end - start, " segundos")
print("Best combination:", best_combination)
```

Resultados: Clasificador logístico

```
_______________logistic_______________
Parametros recibidos:
     holdout logistic 0 0.6
Parametros recibidos:
     holdout logistic 0 1
Parametros recibidos:
     random_subsampling logistic 30 0.6
Parametros recibidos:
     kfold logistic 10 0
Parametros recibidos:
     leaveoneout logistic 10 0
Parametros recibidos:
     stratifiedkfold logistic 10 0
```

SPLIT METHOD	accuracy	error_rate	sensitivity	specifity	precision
holdout	0.754687	0.245312	0.795252	0.709571	0.752809
holdout	0.744215	0.255785	0.750877	0.736559	0.76611
random_subsampling	0.744583	0.255417	0.751826	0.736886	0.768189
kfold	0.739819	0.260181	0.721758	0.70784	0.736778
leaveoneout	0.739837	0.260163	0.749708	0.728495	0.76038
stratifiedkfold	0.733569	0.266431	0.742339	0.722901	0.758112

```
Tiempo de ejecucion: 30.54971680000017  segundos
```

Resultados: Clasificador kneighbors

```
________________kneighbors________________
Parametros recibidos:
     holdout kneighbors 0 0.6
Parametros recibidos:
     holdout kneighbors 0 1
Parametros recibidos:
     random_subsampling kneighbors 30 0.6
Parametros recibidos:
     kfold kneighbors 10 0
Parametros recibidos:
     leaveoneout kneighbors 10 0
Parametros recibidos:
     stratifiedkfold kneighbors 10 0
```

SPLIT METHOD	accuracy	error_rate	sensitivity	specifity	precision
holdout	0.748437	0.251563	0.836795	0.650165	0.726804
holdout	0.817386	0.182614	0.846784	0.783602	0.818079
random_subsampling	0.717083	0.282917	0.77133	0.65658	0.717688
kfold	0.690417	0.309583	0.722759	0.598532	0.66609
leaveoneout	0.72858	0.27142	0.783626	0.665323	0.729053
stratifiedkfold	0.677292	0.322708	0.734268	0.611207	0.690509

```
Tiempo de ejecucion: 22.575431000001117  segundos
```

Resultados: Clasificador DecisionTree

```
________________decisiontree________________
Parametros recibidos:
     holdout decisiontree 0 0.6
Parametros recibidos:
     holdout decisiontree 0 1
Parametros recibidos:
     random_subsampling decisiontree 30 0.6
Parametros recibidos:
     kfold decisiontree 10 0
Parametros recibidos:
     leaveoneout decisiontree 10 0
Parametros recibidos:
     stratifiedkfold decisiontree 10 0
```

SPLIT METHOD	accuracy	error_rate	sensitivity	specifity	precision
holdout	0.734375	0.265625	0.752187	0.713805	0.752187
holdout	1	0	1	1	1
random_subsampling	0.734167	0.265833	0.756457	0.70907	0.750522
kfold	0.664131	0.335869	0.660509	0.626201	0.665206
leaveoneout	0.777986	0.222014	0.797661	0.755376	0.789352
stratifiedkfold	0.651659	0.348341	0.685144	0.612829	0.671621

```
Tiempo de ejecucion: 37.908686800001306  segundos
```

El más lento!

Resultados: Clasificador gaussianNB

```
________________gaussianNB________________
Parametros recibidos:
     holdout gaussianNB 0 0.6
Parametros recibidos:
     holdout gaussianNB 0 1
Parametros recibidos:
     random_subsampling gaussianNB 30 0.6
Parametros recibidos:
     kfold gaussianNB 10 0
Parametros recibidos:
     leaveoneout gaussianNB 10 0
Parametros recibidos:
     stratifiedkfold gaussianNB 10 0
```

SPLIT METHOD	accuracy	error_rate	sensitivity	specifity	precision
holdout	0.723437	0.276562	0.748588	0.692308	0.750708
holdout	0.736085	0.263915	0.745029	0.725806	0.757432
random_subsampling	0.733125	0.266875	0.738667	0.726822	0.758133
kfold	0.719811	0.280189	0.69061	0.6915	0.707238
leaveoneout	0.731707	0.268293	0.74386	0.717742	0.751773
stratifiedkfold	0.718561	0.281439	0.727168	0.708252	0.742979

```
Tiempo de ejecucion: 9.3524352000004  segundos
```

El más rápido!

Resultados: Clasificador SGD (Estocástico)

```
________________sgd________________
Parametros recibidos:
     holdout sgd 0 0.6
Parametros recibidos:
     holdout sgd 0 1
Parametros recibidos:
     random_subsampling sgd 30 0.6
Parametros recibidos:
     kfold sgd 10 0
Parametros recibidos:
     leaveoneout sgd 10 0
Parametros recibidos:
     stratifiedkfold sgd 10 0
```

SPLIT METHOD	accuracy	error_rate	sensitivity	specifity	precision
holdout	0.651563	0.348438	0.645714	0.658621	0.695385
holdout	0.705441	0.294559	0.708772	0.701613	0.731884
random_subsampling	0.692917	0.307083	0.694626	0.691852	0.726202
kfold	0.692893	0.307107	0.673859	0.669532	0.695948
leaveoneout	0.710444	0.289556	0.711111	0.709677	0.737864
stratifiedkfold	0.731065	0.268935	0.789275	0.664	0.744516

```
Tiempo de ejecucion: 30.72785230000045  segundos
```

El que da peores resultados!

¿Cuál es la mejor combinación?

Para escoger la mejor combinación se compararon las métricas de cada clasificador con cada splitter y aquellos que tengan los valores más altos (menos es error) se consideran los mejores.

Best combination: logistic|holdout| k = 0| train_size = 0.6

SPLIT METHOD	accuracy	error_rate	sensitivity	specifity	precision
holdout	0.754687	0.245312	0.795252	0.709571	0.752809
holdout	0.744215	0.255785	0.750877	0.736559	0.76611
random_subsampling	0.744583	0.255417	0.751826	0.736886	0.768189
kfold	0.739819	0.260181	0.721758	0.70784	0.736778
leaveoneout	0.739837	0.260163	0.749708	0.728495	0.76038
stratifiedkfold	0.733569	0.266431	0.742339	0.722901	0.758112

Bibliografía

Decision Tree Classification in Python Tutorial. (2018). Datacamp.com. https://www.datacamp.com/tutorial/decision-tree-classification-python

Sandraviz. (2021). *Decision trees*. Observable. https://observablehq.com/@sandraviz/decision-trees

Sklearn.Model_selection.Cross_validate — documentación de scikit-learn - 0.24.1. (s/f). Github.Io. https://qu4nt.github.io/sklearn-doc-es/modules/generated/sklearn.model_selection.cross_validate.html

Sklearn.Model_selection.Cross_val_predict. (s/f). Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_predict.html

Sklearn.tree.decisionTreeClassifier. (s/f). Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Tyagi, N. (s/f). *Information gain, Gini index, entropy and gain ratio in decision trees*. Analyticssteps.com. https://www.analyticssteps.com/blogs/what-gini-index-and-information-gain-decision-trees