Arbres de décision

March 16, 2024

0.0.1 Introduction

0.0.2 Qu'est-ce qu'un arbre de décision ?

Un arbre de décision est un diagramme de flux ramifié ou un arbre. Il comprend les composants suivants :

- Une variable cible (label)
- Un nœud racine : c'est le nœud qui commence le processus de fractionnement en trouvant la variable qui fractionne le mieux la variable cible
- Pureté du nœud : les nœuds de décision sont généralement impurs, ou un mélange des deux classes de la variable cible (0,1). Les nœuds purs sont ceux qui ont une classe - d'où le terme pur. Ils ont soit des points verts ou rouges uniquement dans l'image.
- Nœuds de décision : ce sont des nœuds suivants ou intermédiaires, où la variable cible est à nouveau divisée par d'autres variables
- Les nœuds feuilles ou les nœuds terminaux sont des nœuds purs, ils sont donc utilisés pour faire une prédiction d'un nombre ou d'une classe.

$$Gini = 1 - \sum_{i=1}^{j} P(i)^2$$

En général, un arbre de décision prend une déclaration ou une hypothèse ou une condition, puis décide si la condition est vérifiée ou non. Les conditions sont affichées le long des branches et le résultat de la condition, tel qu'appliqué à la variable cible, est affiché sur le nœud.

$$Gini = 1 - \sum_{i=1}^{j} P(i)^2$$

Les arbres de décision offrent une grande flexibilité dans la mesure où nous pouvons utiliser à la fois des variables numériques et catégorielles pour diviser les données cibles. Les données catégorielles sont réparties selon les différentes classes de la variable. Le numérique est un peu plus délicat car nous devons nous diviser en seuils pour la condition testée, tels que <18 et 18, par exemple. Une variable numérique peut apparaître plusieurs fois dans les données avec des coupures ou des seuils différents. Les classements finaux peuvent également être répétés.

0.0.3 Utilisation d' arbres de decision pour la data science

1. Flux d'informations à travers l'arbre de décision
2. Comment les arbres de décision sélectionnent-ils la variable à diviser aux nœuds de décision ?
3. Comment décide-t-il que l'arbre a suffisamment de branches et qu'il doit cesser de se fendre ?

Approche descendante gourmande (Greedy)

Les arbres de décision suivent une approche descendante et gourmande connue sous le nom de fractionnement binaire récursif. L'approche de fractionnement binaire récursif est descendante car elle commence au sommet de l'arbre, puis divise successivement l'espace des prédicteurs. À chaque division, l'espace des prédicteurs est divisé en 2 et est affiché via deux nouvelles branches pointant vers le bas. L'algorithme est dit gourmand car à chaque étape du processus, la meilleure répartition est faite pour cette étape. Il ne se projette pas vers l'avant et essaie de choisir une division qui pourrait être plus optimale pour l'arbre global.

L'algorithme évalue donc toutes les variables sur certains critères statistiques puis choisit la variable qui performe le mieux sur les critères.

Critère de sélection variable

C'est là que réside la véritable complexité et sophistication de la décision. Les variables sont sélectionnées sur un critère statistique complexe qui est appliqué à chaque nœud de décision. Désormais, le critère de sélection des variables dans les arbres de décision peut être effectué via deux approches :

1. Entropie et gain d'information
2. Indice de Gini

Les deux critères sont globalement similaires et cherchent à déterminer quelle variable diviserait les données pour que les nœuds enfants sous-jacents soient les plus homogènes ou les plus purs. Les deux sont utilisés dans différents algorithmes d'arbre de décision. Pour ajouter à la confusion, il n'est pas clair quelle est l'approche préférée. Donc, il faut avoir une compréhension des deux.

1- Qu'est-ce que l'Entropie ? Dans le contexte des arbres de décision, l'entropie est une mesure du désordre ou de l'impureté dans un nœud. Ainsi, un nœud avec une composition plus variable, tel que 2Pass et 2 Fail, serait considéré comme ayant une entropie plus élevée qu'un nœud qui n'a que réussi ou échoué. Le niveau maximum d'entropie ou de désordre est donné par 1 et l'entropie minimum est donnée par une valeur 0.

Les nœuds feuilles dont toutes les instances appartiennent à 1 classe auraient une entropie de 0. Alors que l'entropie d'un nœud où les classes sont divisées de manière égale serait de 1.

$$Gini = 1 - \sum_{i=1}^{j} P(i)^2$$

L'entropie est mesurée par la formule :
Où le pi est la probabilité de sélectionner au hasard un exemple dans la classe i.

Maintenant, essentiellement, ce qu'un arbre de décision fait pour déterminer le nœud racine est de calculer l'entropie pour chaque variable et ses divisions potentielles. Pour cela, nous devons calculer une séparation potentielle de chaque variable, calculer l'entropie moyenne sur les deux ou tous les nœuds, puis le changement d'entropie vis-à-vis du nœud parent. Ce changement d'entropie est appelé gain d'information et

représente la quantité d'informations qu'une caractéristique fournit pour la variable cible.

$$Gini = 1 - \sum_{i=1}^{j} P(i)^2$$

2- Indice de Gini

L'autre façon de diviser un arbre de décision est via l'indice de Gini. La méthode Entropy and Information Gain se concentre sur la pureté et l'impureté dans un nœud. L'indice de Gini ou impureté mesure la probabilité qu'une instance aléatoire soit mal classée lorsqu'elle est choisie au hasard. Plus l'indice de Gini est bas, plus la probabilité d'erreur de classification est faible.

$$Gini = 1 - \sum_{i=1}^{j} P(i)^2$$

La formule de l'indice de Gini:
Où j représente le nombre de classes dans la variable cible — Réussite et Échec par exemple

P(i) représente le ratio Pass/Total nombre d'observations dans le nœud.

Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklea

Quand arrêter le fractionnement (splitting) ? Il existe de nombreuses façons de résoudre ce problème grâce au réglage des hyperparamètres. Nous pouvons définir la profondeur maximale de notre arbre de décision à l'aide du paramètre max_depth. Plus la valeur de max_depth est élevée, plus votre arbre sera complexe. L'erreur d'entraînement diminuera hors cours si nous augmentons la valeur max_depth mais lorsque nos données de test entreront en jeu, nous obtiendrons une très mauvaise précision. Par conséquent, vous avez besoin d'une valeur qui ne sur-adaptera ni ne sous-adaptera nos données et pour cela, vous pouvez utiliser GridSearchCV.

Une autre méthode consiste à définir le nombre minimum d'échantillons pour chaque déversement. Il est noté min_samples_split. Ici, nous spécifions le nombre minimum d'échantillons requis pour faire un déversement. Par exemple, nous pouvons utiliser un minimum de 10 échantillons pour prendre une décision. Cela signifie que si un nœud a moins de 10 échantillons, en utilisant ce paramètre, nous pouvons arrêter la division supplémentaire de ce nœud et en faire un nœud feuille.

Il existe d'autres hyperparamètres tels que :

min_samples_leaf - représente le nombre minimum d'échantillons requis pour être dans le nœud feuille. Plus vous augmentez le nombre, plus la possibilité de surajustement est grande. max_features - cela nous aide à décider du nombre de fonctionnalités à prendre en compte lors de la recherche de la meilleure répartition.

Pruning

C'est une autre méthode qui peut nous aider à éviter le surajustement. Il aide à améliorer les performances de l'arbre en coupant les nœuds ou sous-nœuds qui ne sont pas significatifs. Il supprime les branches qui ont une très faible importance.

Il existe principalement 2 façons de tailler :

Pré-taille - nous pouvons arrêter de faire pousser l'arbre plus tôt, ce qui signifie que nous pouvons tailler/supprimer/couper un nœud s'il a une faible importance lors de la croissance de l'arbre.

Post-élagage - une fois que notre arbre est construit à sa profondeur, nous pouvons commencer à élaguer les nœuds en fonction de leur importance.

Implementation

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
df = pd.read_csv('titanic.csv')
```

[3]:
```
df.columns
```

[3]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

[4]:
```
df = df[['Survived','Pclass','Sex','Age','Fare']]
```

[5]:
```
df.head()
```

[5]:

	Survived	Pclass	Sex	Age	Fare
0	0	3	male	22.0	7.2500
1	1	1	female	38.0	71.2833
2	1	3	female	26.0	7.9250
3	1	1	female	35.0	53.1000
4	0	3	male	35.0	8.0500

[6]:
```
df.isnull().sum()
```

[6]:
```
Survived      0
Pclass        0
Sex           0
Age         177
Fare          0
dtype: int64
```

[7]:
```
df['Age'] = df['Age'].fillna(np.mean(df['Age']))
```

[8]:
```
df.isnull().sum()
```

[8]:
```
Survived    0
Pclass      0
Sex         0
Age         0
Fare        0
dtype: int64
```

[9]:
```
#Changer la colonne de catégorie en colonne numérique
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

[10]:
```
df['Sex'] = le.fit_transform(df.Sex)
```

[11]:
```
df['Sex'].value_counts()
```

[11]:
```
Sex
1    577
0    314
Name: count, dtype: int64
```

[12]:
```
X = df[['Pclass','Sex','Age','Fare']]
y= df['Survived']
```

[13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[14]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn import tree


clf = DecisionTreeClassifier(random_state=42,criterion='gini')
clf.fit(X_train,y_train)
```

[14]:
```
DecisionTreeClassifier(random_state=42)
```

[15]:
```
# Training accuracy
predictions_train = clf.predict(X_train)
accuracy_score(y_train,predictions_train)
```

[15]:
```
0.9791332263242376
```

[16]:
```
# Test accuracy
predictions_test=clf.predict(X_test)
accuracy_score(y_test, predictions_test)
```

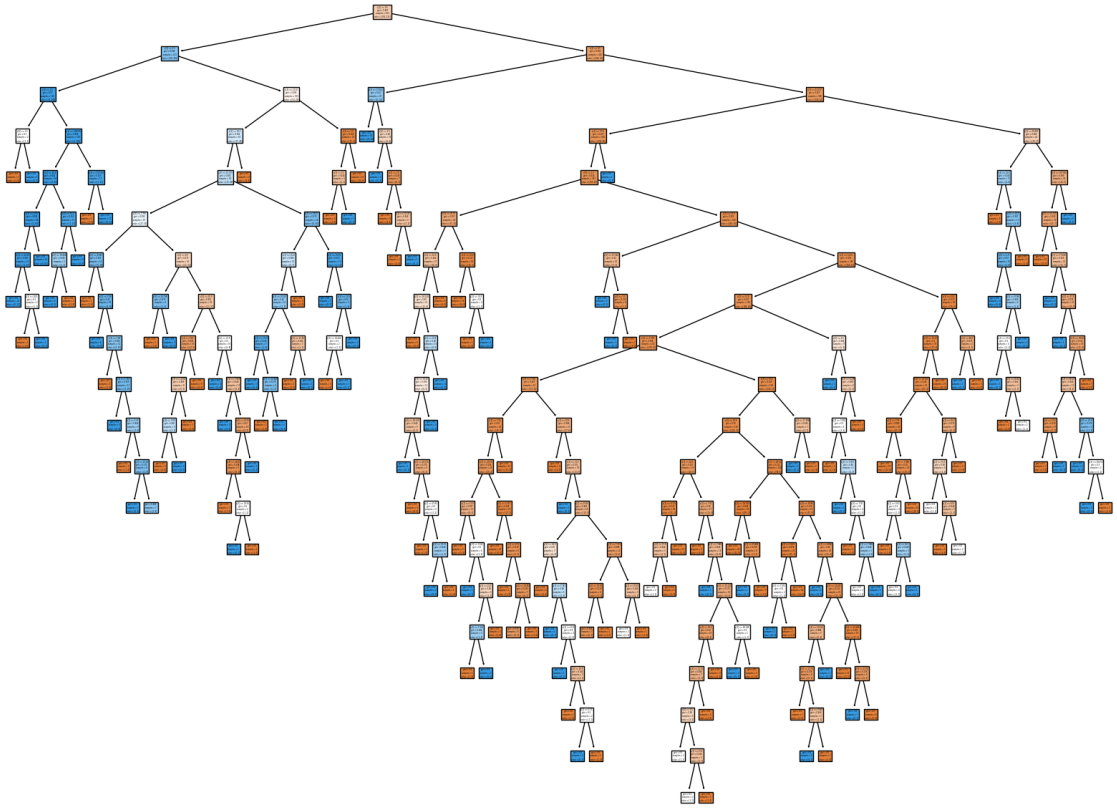
[16]:
```
0.75
```

[17]:
```
print(confusion_matrix(y_test,predictions_test))
```

```
[[126  31]
 [ 36  75]]
```

[ ]:

[18]:
```
# Visualisation de notre arbre de décision final
plt.figure(figsize=(20,15))
tree.plot_tree(clf,filled=True)
plt.show()
```



Élagage de notre arbre de décision

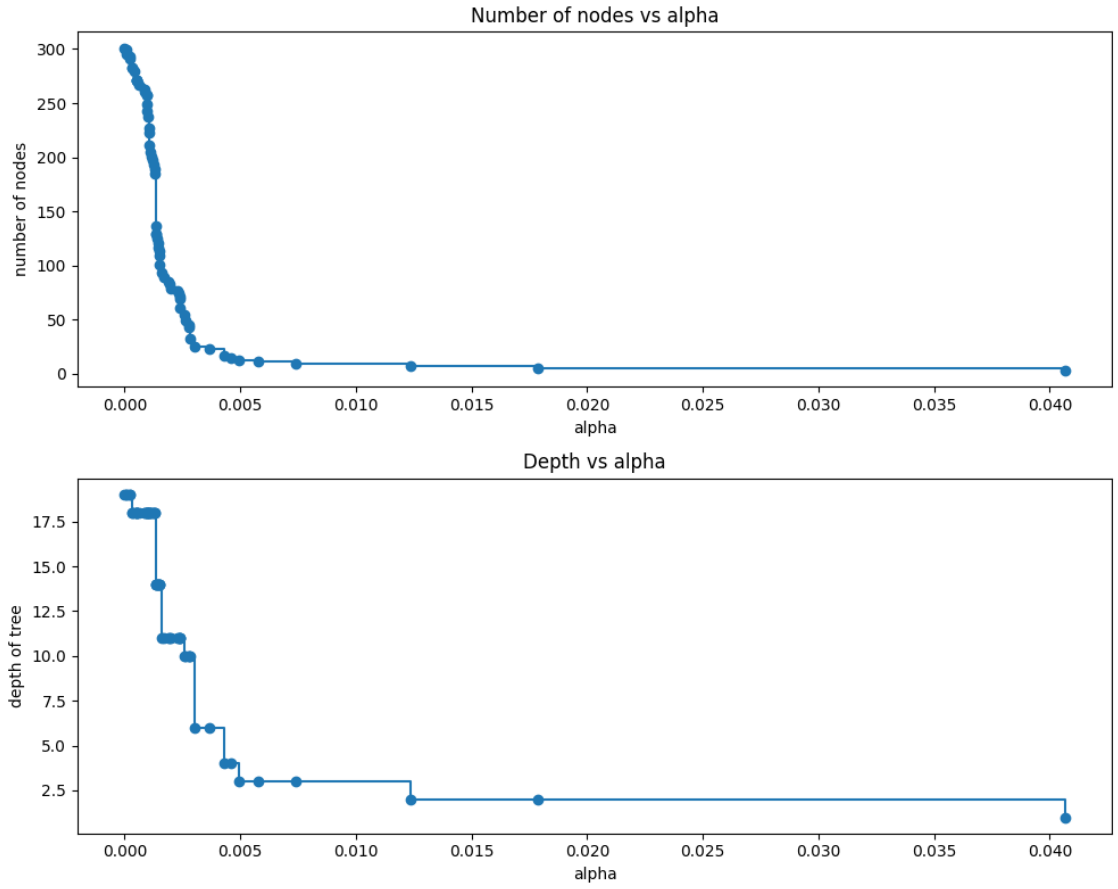
[19]:
```
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

[20]:
```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print("Number of nodes in the last tree is: {} with ccp_alpha: {}".
  ↪format(clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.13235809539273619
```

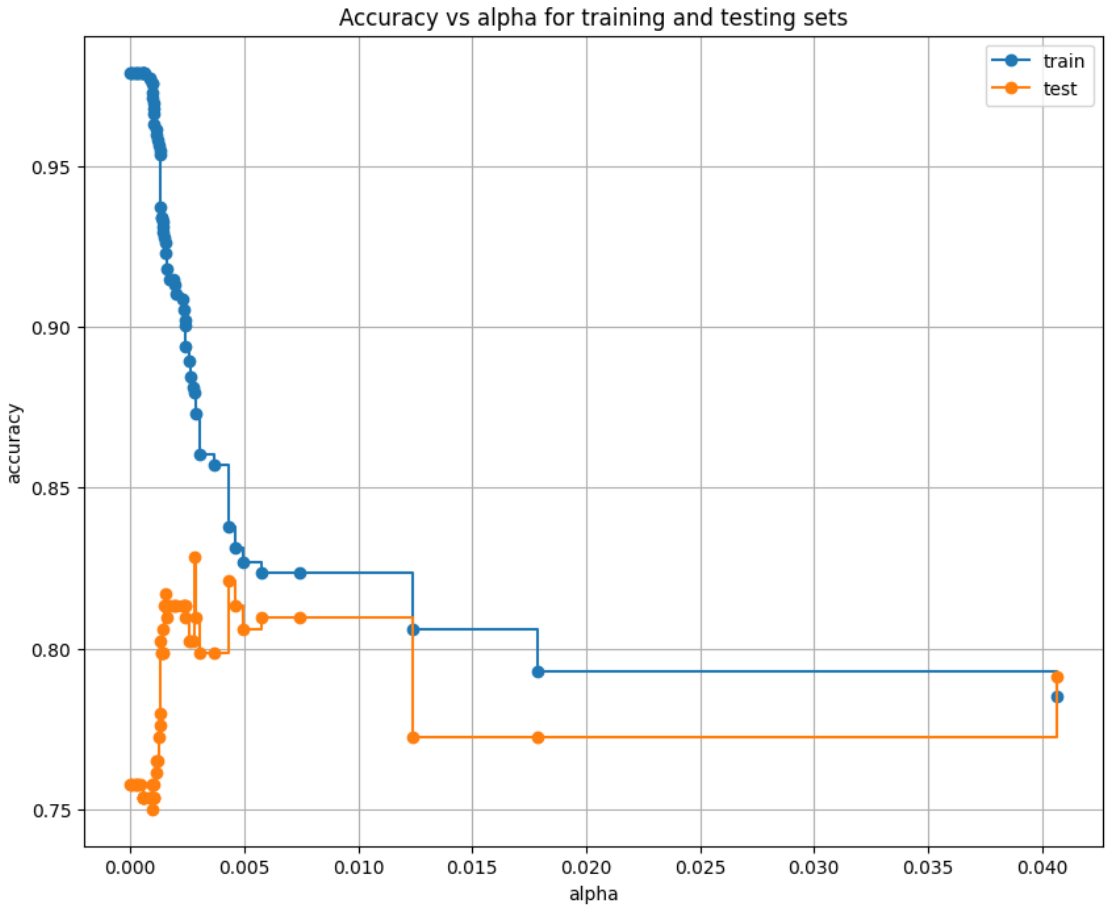
[21]:
```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]
fig, ax = plt.subplots(2, 1,figsize=(10,8))
ax[0].plot(ccp_alphas, node_counts, marker='o', drawstyle="steps-post")
ax[0].set_xlabel("alpha")
ax[0].set_ylabel("number of nodes")
ax[0].set_title("Number of nodes vs alpha")
ax[1].plot(ccp_alphas, depth, marker='o', drawstyle="steps-post")
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
fig.tight_layout()
```



[22]:
```
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]
```

```
fig, ax = plt.subplots(figsize=(10,8))
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o',␣
 ↪label="train",drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o',␣
 ↪label="test",drawstyle="steps-post")
ax.legend()
plt.grid()
plt.show()
```



Précision après la reduction de l'arbre

[23]:
```
clf = DecisionTreeClassifier(random_state=0, ccp_alpha=0.008)
clf.fit(X_train,y_train)
```

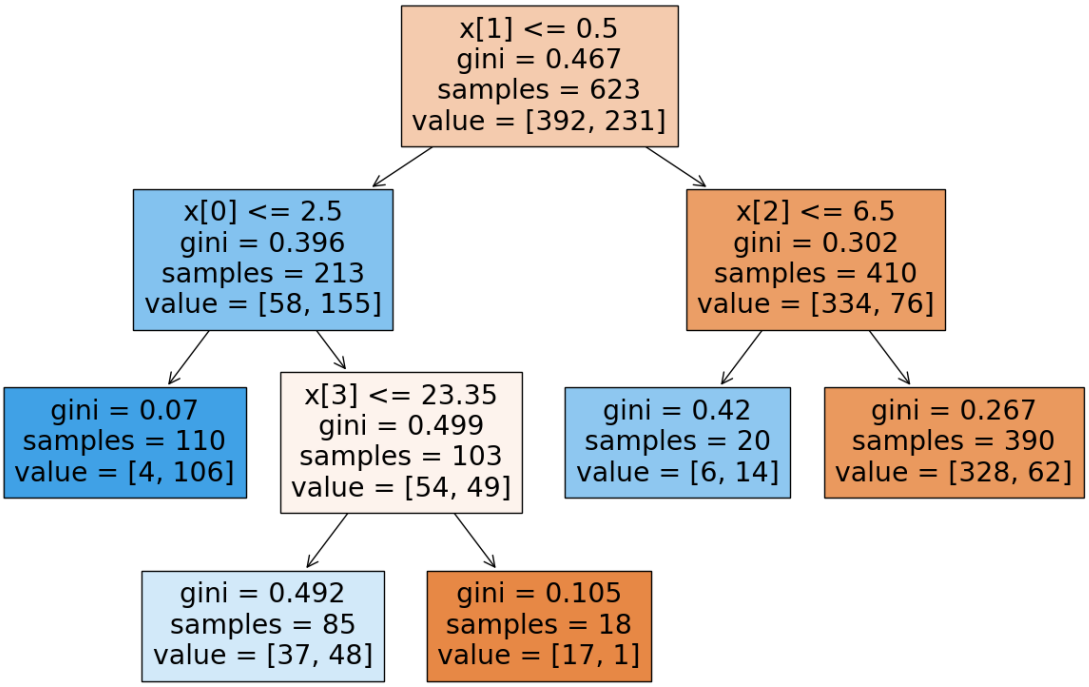
[23]:
```
DecisionTreeClassifier(ccp_alpha=0.008, random_state=0)
```

[24]:
```
pred=clf.predict(X_test)
accuracy_score(y_test, pred)
```

[24]:
```
0.8097014925373134
```

[25]:
```
plt.figure(figsize=(15,10))
tree.plot_tree(clf,filled=True)
```

[25]:
```
[Text(0.5, 0.875, 'x[1] <= 0.5\ngini = 0.467\nsamples = 623\nvalue = [392,
231]'),
 Text(0.25, 0.625, 'x[0] <= 2.5\ngini = 0.396\nsamples = 213\nvalue = [58,
155]'),
 Text(0.125, 0.375, 'gini = 0.07\nsamples = 110\nvalue = [4, 106]'),
 Text(0.375, 0.375, 'x[3] <= 23.35\ngini = 0.499\nsamples = 103\nvalue = [54,
49]'),
 Text(0.25, 0.125, 'gini = 0.492\nsamples = 85\nvalue = [37, 48]'),
 Text(0.5, 0.125, 'gini = 0.105\nsamples = 18\nvalue = [17, 1]'),
 Text(0.75, 0.625, 'x[2] <= 6.5\ngini = 0.302\nsamples = 410\nvalue = [334,
76]'),
 Text(0.625, 0.375, 'gini = 0.42\nsamples = 20\nvalue = [6, 14]'),
 Text(0.875, 0.375, 'gini = 0.267\nsamples = 390\nvalue = [328, 62]')]
```



[26]:
```
X.describe()
```

[26]:

	Pclass	Sex	Age	Fare
count	891.000000	891.000000	891.000000	891.000000
mean	2.308642	0.647587	29.699118	32.204208
std	0.836071	0.477990	13.002015	49.693429
min	1.000000	0.000000	0.420000	0.000000
25%	2.000000	0.000000	22.000000	7.910400
50%	3.000000	1.000000	29.699118	14.454200
75%	3.000000	1.000000	35.000000	31.000000
max	3.000000	1.000000	80.000000	512.329200

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: