tarefa

February 5, 2021

0.1 Datasets utilizados

0.1.1 Dataset 1 https://archive.ics.uci.edu/ml/machine-learning-databases/glass/

0.1.2 Dataset 2 https://archive.ics.uci.edu/ml/machine-learning-databases/blood-transfusion/

0.1.3 Dataset 3 https://archive.ics.uci.edu/ml/datasets/banknote+authentication

Outros datasets

https://archive.ics.uci.edu/ml/datasets/Cervical+Cancer+Behavior+Risk

https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records

https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/

[57]:
```
import numpy as np
from sklearn.datasets import load_iris

iris = load_iris()
dataset1 = np.loadtxt('datasets/glass.data', delimiter=',')
dataset2 = np.loadtxt('datasets/transfusion.data', delimiter=',', skiprows=1)
dataset3 = np.loadtxt('datasets/data_banknote_authentication.txt',␣
 ↪delimiter=',')
```

[58]:
```
Xi, yi = iris.data[:,2:], iris.target      # Dataset Iris utilizado nos vídeos
X1, y1 = dataset1[:,4:6], dataset1[:,-1]   # Dataset 1
X2, y2 = dataset2[:,2:4], dataset2[:,-1]   # Dataset 2
X3, y3 = dataset3[:,:2], dataset3[:,-1]    # Dataset 3

print(f"Dataset Iris: {Xi.shape}, {yi.shape}, {set(yi)}\n")
print(f"Dataset1: {X1.shape}, {y1.shape}, {set(y1)}\n")
print(f"Dataset2: {X2.shape}, {y2.shape}, {set(y2)}\n")
print(f"Dataset3: {X3.shape}, {y3.shape}, {set(y3)}")
```

```
Dataset Iris: (150, 2), (150,), {0, 1, 2}

Dataset1: (214, 2), (214,), {1.0, 2.0, 3.0, 5.0, 6.0, 7.0}

Dataset2: (748, 2), (748,), {0.0, 1.0}
```

```
Dataset3: (1372, 2), (1372,), {0.0, 1.0}
```

0.2 Árvore da tarefa

0.2.1 Heurística:

Melhor valor da característica: é o valor mais próximo da média da característica

Melhor característica: é a característica com o melhor valor mais próximo da média do y

[59]:
```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_validate
import numpy as np
from collections import Counter
from sklearn.base import BaseEstimator, ClassifierMixin


# Valor mais próximo da média da característica
def melhorValorTarefa(x, y):
    valor = None
    impValor = None
    xmean = np.mean(x) # média da característica
    valor = x[(x - xmean).argmin()] # valor mais próximo da média
    return valor


# Característica com melhor valor mais próximo da média do y
def melhorCaracteristicaTarefa(X, y):
    impurezas = []
    valores = []
    for caracteristica in range(X.shape[1]):
        valor = melhorValorTarefa(X[:,caracteristica], y)
        valores.append(valor)

    ymean = np.mean(y) #média do y
    caracteristica = (valores - ymean).argmin() # Característica com melhor␣
 ↪valor mais próximo da média do y

    return caracteristica, valores[caracteristica]

def maisFrequenteTarefa(y):
    return Counter(y.flat).most_common(1)[0][0]


class ArvoreTarefa(BaseEstimator, ClassifierMixin):
    def fit(self, X, y):
        self.caracteristica, self.valor = melhorCaracteristicaTarefa(X, y)
```

```
        maiores = X[:,self.caracteristica] > self.valor
        if sum(maiores)>0 and sum(~maiores)>0:
            self.maiores = ArvoreTarefa()
            self.maiores.fit(X[maiores,:],y[maiores])
            self.menores = ArvoreTarefa()
            self.menores.fit(X[~maiores,:],y[~maiores])
        else:
            self.resposta = maisFrequenteTarefa(y)
    def predict(self, X):
        y = np.empty((X.shape[0]))
        if hasattr(self, 'resposta'):
            y[:] = self.resposta
        else:
            maiores = X[:,self.caracteristica] > self.valor
            y[maiores] = self.maiores.predict(X[maiores,:])
            y[~maiores] = self.menores.predict(X[~maiores,:])
        return y
```

0.3 Árvore dos vídeos

[60]:
```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_validate
import numpy as np
from collections import Counter
from sklearn.base import BaseEstimator, ClassifierMixin

def impureza(y): #Gini
    labels = list(set(y))
    labels.sort()
    probabilidades = np.zeros((len(labels),))
    for i,k in enumerate(labels):
        probabilidades[i] = sum(y==k)/len(y)
    result = 1 - sum(probabilidades**2)
    return result


def impurezaValor(x, y, valor):
    maiores = x>valor
    impurezaMaiores = impureza(y[maiores])
    propMaiores = sum(maiores)/len(y)
    impurezaMenores = impureza(y[~maiores])
    propMenores = sum(~maiores)/len(y)
    impurezaTotal = propMaiores*impurezaMaiores + propMenores*impurezaMenores
    return impurezaTotal, impurezaMaiores, impurezaMenores


def melhorValor(x, y):
```

```
    result = None
    menorImpureza = float('inf')
    xmax = np.max(x)
    xmin = np.min(x)
    while True:
        valor = (xmin+xmax)/2
        impTotal, impMaiores, impMenores = impurezaValor(x, y, valor)
        if impTotal < menorImpureza:
            menorImpureza = impTotal
            result = valor
            if impMaiores == 0 or impMenores == 0:
                break
            if impMaiores < impMenores:
                xmin = valor
            else:
                xmax = valor
        else:
            break
    return result, menorImpureza


def melhorCaracteristica(X, y):
    impurezas = []
    valores = []
    for caracteristica in range(X.shape[1]):
        valor, imp = melhorValor(X[:,caracteristica], y)
        impurezas.append(imp)
        valores.append(valor)
    impurezas = np.array(impurezas)
    caracteristica = np.argmin(impurezas)
    return caracteristica, valores[caracteristica], impurezas[caracteristica]


def maisFrequente(y):
    return Counter(y.flat).most_common(1)[0][0]


class ArvoreVideo(BaseEstimator, ClassifierMixin):
    def fit(self, X, y):
        self.caracteristica, self.valor, self.imp = melhorCaracteristica(X,y)
        maiores = X[:,self.caracteristica] > self.valor
        if sum(maiores)>0 and sum(~maiores)>0:
            self.maiores = ArvoreVideo()
            self.maiores.fit(X[maiores,:],y[maiores])
            self.menores = ArvoreVideo()
            self.menores.fit(X[~maiores,:],y[~maiores])
        else:
```

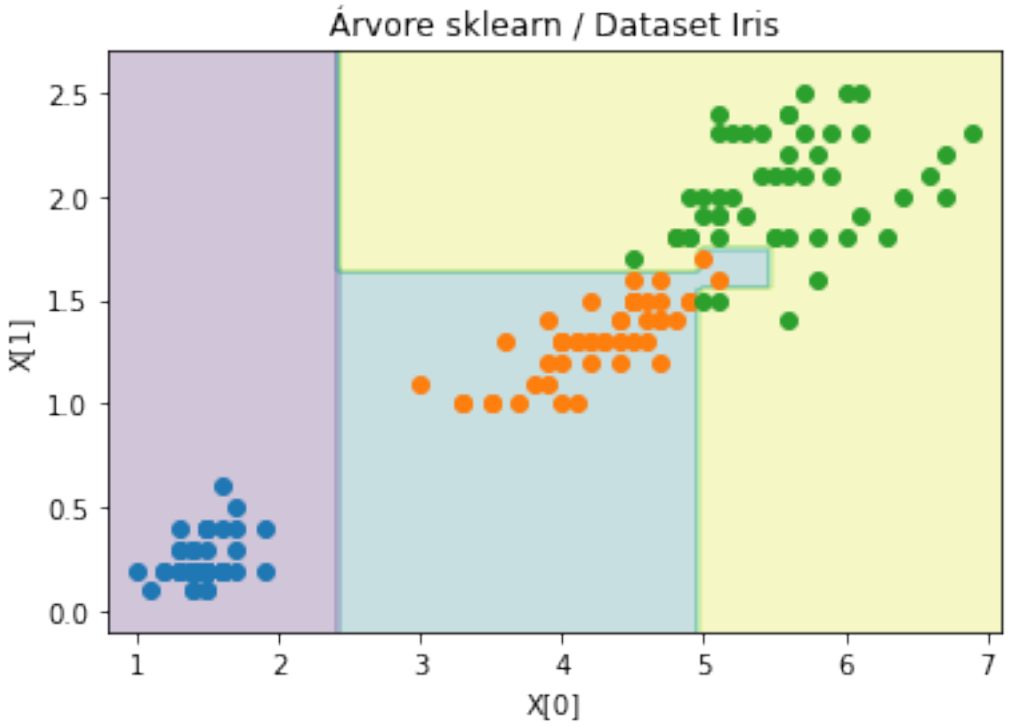
```
            self.resposta = maisFrequente(y)
    def predict(self, X):
        y = np.empty((X.shape[0]))
        if hasattr(self, 'resposta'):
            y[:] = self.resposta
        else:
            maiores = X[:,self.caracteristica] > self.valor
            y[maiores] = self.maiores.predict(X[maiores,:])
            y[~maiores] = self.menores.predict(X[~maiores,:])
        return y
```

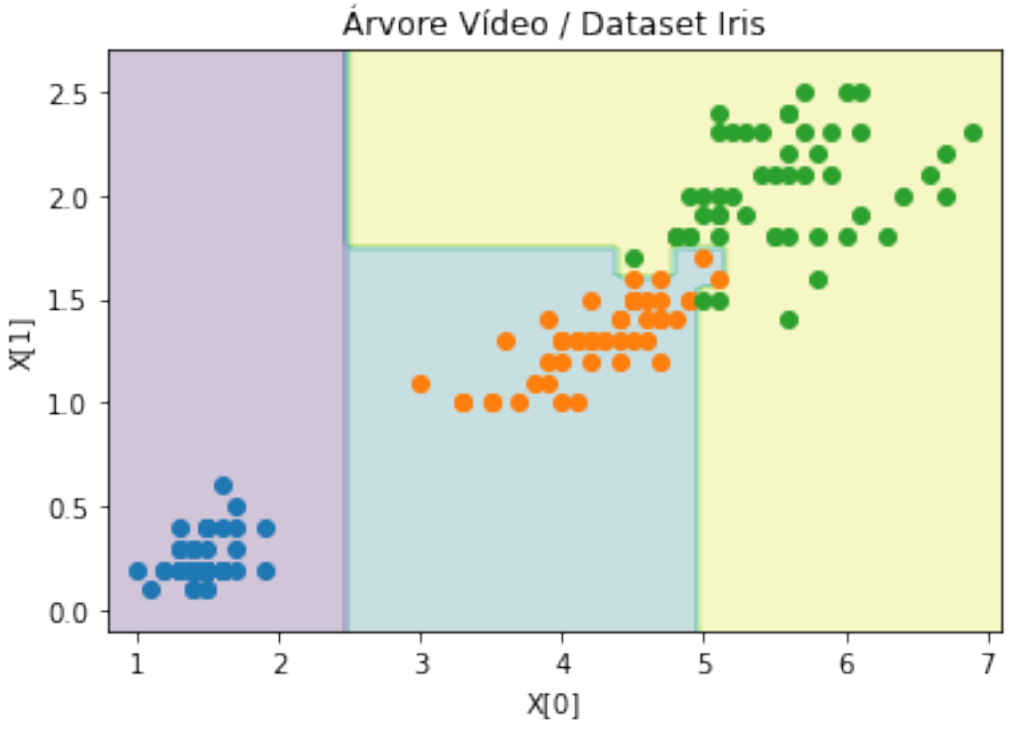
0.4 Plot região de decisão

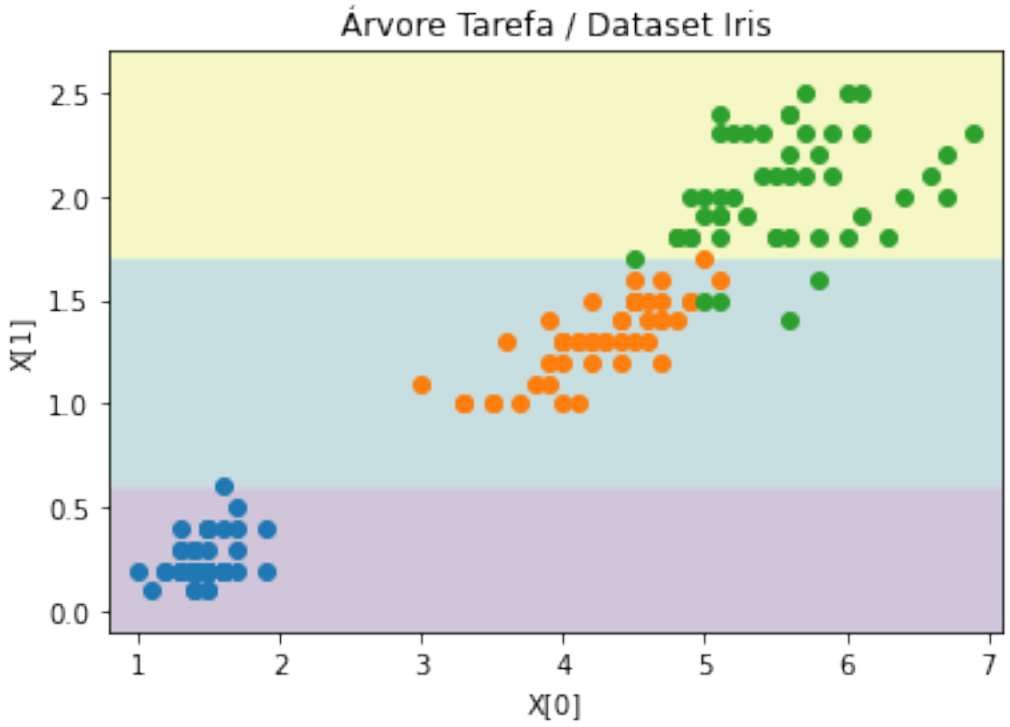
[61]:
```
import matplotlib.pyplot as plt

def plotDecisao(modelo, X, y, nome_modelo, titulo):
    modelo.fit(X, y)
    x0s = np.linspace(np.min(X[:,0])-0.2, np.max(X[:,0])+0.2, 100)
    x1s = np.linspace(np.min(X[:,1])-0.2, np.max(X[:,1])+0.2, 100)
    x0, x1 = np.meshgrid(x0s, x1s)
    Xdec = np.c_[x0.ravel(), x1.ravel()]
    ypred = modelo.predict(Xdec)
    plt.contourf(x0, x1, ypred.reshape(x0.shape), alpha=0.25)
    for k in set(y):
        plt.plot(X[:,0][y==k], X[:,1][y==k], 'o')
    plt.xlabel(f"X[0]")
    plt.ylabel(f"X[1]")
    plt.title(f"{nome_modelo} / {titulo}")
    plt.show()
```

0.5 Comparações das regiões de decisão para o dataset Iris do sklearn

[62]:
```
from sklearn.tree import DecisionTreeClassifier

plotDecisao(DecisionTreeClassifier(), Xi, yi, "Árvore sklearn", "Dataset Iris")
plotDecisao(ArvoreVideo(), Xi, yi, "Árvore Vídeo", "Dataset Iris")
plotDecisao(ArvoreTarefa(), Xi, yi, "Árvore Tarefa", "Dataset Iris")
```


Árvore sklearn / Dataset Iris
X[1]
X[0]
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7


Árvore Vídeo / Dataset Iris
X[1]
X[0]
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7



0.6 Comparações da validação cruzada com outros 03 (três) datasets

[63]:
```
from sklearn.tree import DecisionTreeClassifier

#Dataset Iris
print("\nCross Validade Dataset Iris:")
scoresSklean = cross_validate(DecisionTreeClassifier(), Xi, yi)
scoresVideo = cross_validate(ArvoreVideo(), Xi, yi)
scoresTarefa = cross_validate(ArvoreTarefa(), Xi, yi)

print(f"\tSklearn Test Score: {scoresSklean['test_score']}")
print(f"\tSklean Mean Test Score: {np.mean(scoresSklean['test_score'])}\n")

print(f"\tVideo Test Score: {scoresVideo['test_score']}")
print(f"\tvideo Mean Test Score: {np.mean(scoresVideo['test_score'])}\n")

print(f"\tTarefa Test Score: {scoresTarefa['test_score']}")
print(f"\tTarefa Test Score: {np.mean(scoresTarefa['test_score'])}\n")


#Dataset 1
print("\nCross Validade Dataset 1:")
scoresSklean = cross_validate(DecisionTreeClassifier(), X1, y1)
```

```
scoresVideo = cross_validate(ArvoreVideo(), X1, y1)
scoresTarefa = cross_validate(ArvoreTarefa(), X1, y1)

print(f"\tSklearn Test Score: {scoresSklean['test_score']}")
print(f"\tSklearn Mean Test Score: {np.mean(scoresSklean['test_score'])}\n")

print(f"\tVideo Test Score: {scoresVideo['test_score']}")
print(f"\tVideo Mean Test Score: {np.mean(scoresVideo['test_score'])}\n")

print(f"\tTarefa Test Score: {scoresTarefa['test_score']}")
print(f"\tTarefa Mean Test Score: {np.mean(scoresTarefa['test_score'])}\n")


#Dataset 2
print("\nCross Validade Dataset 2:")
scoresSklean = cross_validate(DecisionTreeClassifier(), X2, y2)
scoresVideo = cross_validate(ArvoreVideo(), X2, y2)
scoresTarefa = cross_validate(ArvoreTarefa(), X2, y2)

print(f"\tSklearn Test Score: {scoresSklean['test_score']}")
print(f"\tSklearn Mean Test Score: {np.mean(scoresSklean['test_score'])}\n")

print(f"\tVideo Test Score: {scoresVideo['test_score']}")
print(f"\tVideo Mean Test Score: {np.mean(scoresVideo['test_score'])}\n")

print(f"\tTarefa Test Score: {scoresTarefa['test_score']}")
print(f"\tTarefa Mean Test Score: {np.mean(scoresTarefa['test_score'])}\n")


#Dataset 3
print("\nCross Validade Dataset 3:")
scoresSklean = cross_validate(DecisionTreeClassifier(), X3, y3)
scoresVideo = cross_validate(ArvoreVideo(), X3, y3)
scoresTarefa = cross_validate(ArvoreTarefa(), X3, y3)

print(f"\tSklearn Test Score: {scoresSklean['test_score']}")
print(f"\tSklean Mean Test Score: {np.mean(scoresSklean['test_score'])}\n")

print(f"\tVideo Test Score: {scoresVideo['test_score']}")
print(f"\tVideo Mean Test Score: {np.mean(scoresVideo['test_score'])}\n")

print(f"\tTarefa Test Score: {scoresTarefa['test_score']}")
print(f"\tTarefa Mean Test Score: {np.mean(scoresTarefa['test_score'])}\n")
```

```
Cross Validade Dataset Iris:
        Sklearn Test Score: [0.96666667 0.96666667 0.9        0.93333333 1.
]
```

```
        Sklean Mean Test Score: 0.9533333333333334

        Video Test Score: [0.96666667 0.96666667 0.9        0.93333333 1.
]
        video Mean Test Score: 0.9533333333333334

        Tarefa Test Score: [0.96666667 0.96666667 0.9        0.93333333
0.96666667]
        Tarefa Test Score: 0.9466666666666667


Cross Validade Dataset 1:
        Sklearn Test Score: [0.39534884 0.58139535 0.44186047 0.58139535
0.4047619 ]
        Sklearn Mean Test Score: 0.4809523809523809

        Video Test Score: [0.30232558 0.48837209 0.46511628 0.51162791 0.5
]
        Video Mean Test Score: 0.4534883720930233

        Tarefa Test Score: [0.41860465 0.58139535 0.46511628 0.58139535
0.4047619 ]
        Tarefa Mean Test Score: 0.4902547065337763


Cross Validade Dataset 2:
        Sklearn Test Score: [0.55333333 0.69333333 0.72       0.73154362
0.73154362]
        Sklearn Mean Test Score: 0.6859507829977629

        Video Test Score: [0.52666667 0.68       0.73333333 0.7114094
0.69127517]
        Video Mean Test Score: 0.6685369127516779

        Tarefa Test Score: [0.74666667 0.72666667 0.73333333 0.76510067
0.73154362]
        Tarefa Mean Test Score: 0.740662192393736


Cross Validade Dataset 3:
        Sklearn Test Score: [0.96       0.92363636 0.9270073  0.94160584
0.9270073 ]
        Sklean Mean Test Score: 0.9358513603185136

        Video Test Score: [0.95636364 0.89090909 0.91970803 0.94525547
0.92335766]
        Video Mean Test Score: 0.9271187790311878
```

```
        Tarefa Test Score: [0.78545455 0.71272727 0.73722628 0.79927007
0.77372263]
        Tarefa Mean Test Score: 0.7616801592568015
```

0.7 Interpretação da tarefa

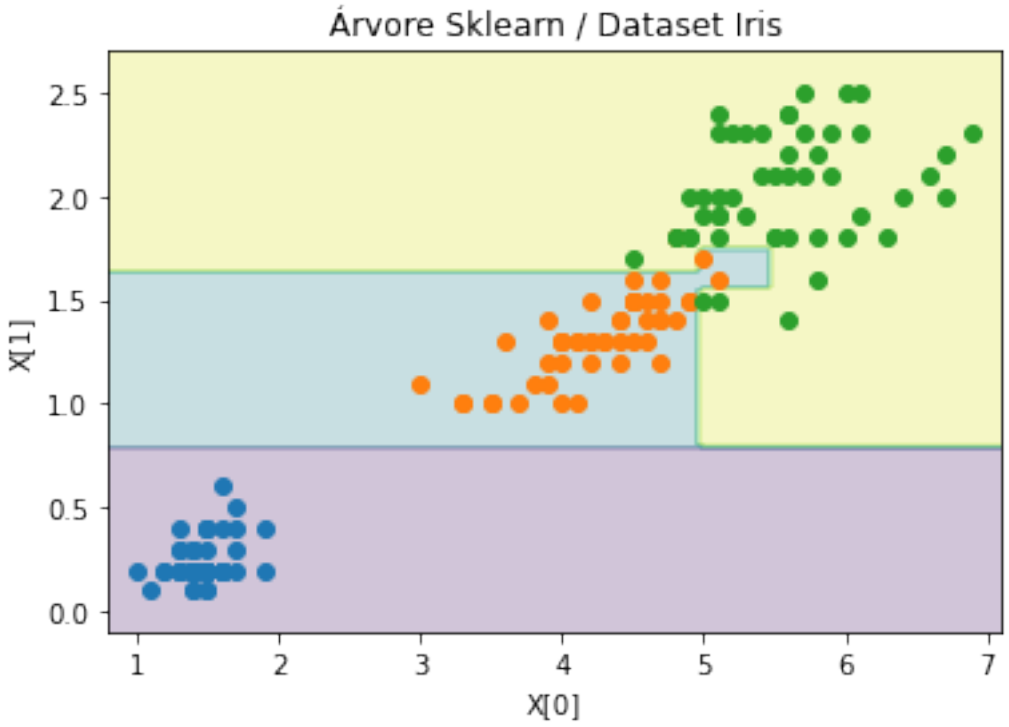
Foram observados os seguintes pontos no desenvolvimento da tarefa:

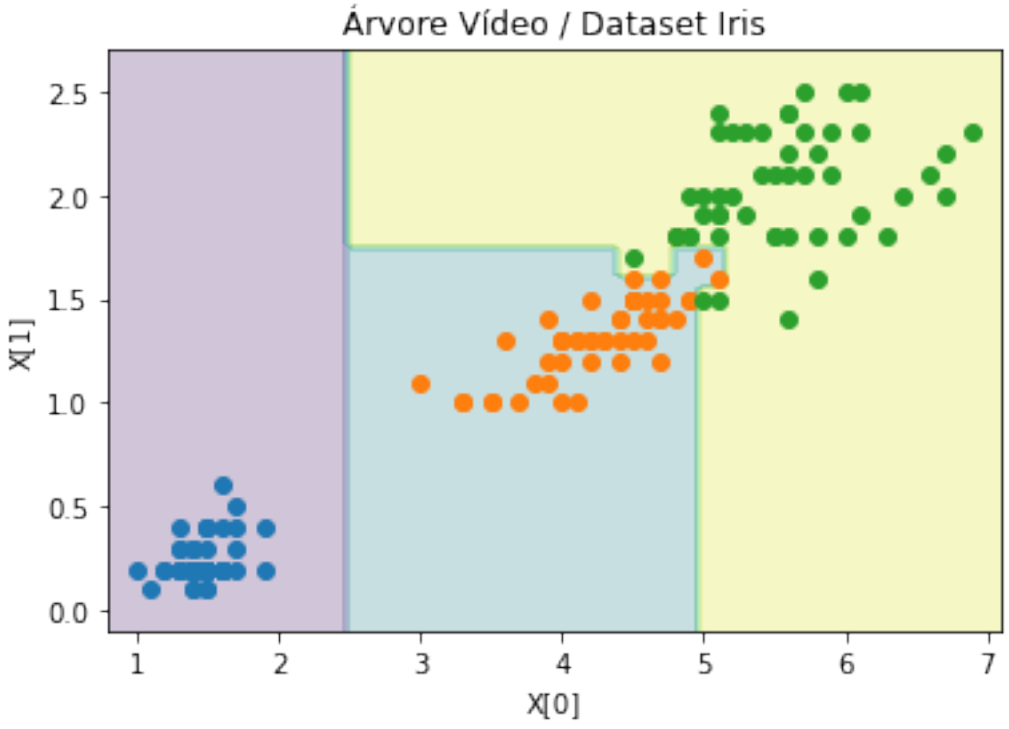
1. Por se tratar de uma heurística para escolha da melhor característica, o conhecimento da base influencia diretamente nos resultados;
2. Cada característica influencia de uma forma os resultados, visto que, no dataset 2, a heurística da tarefa foi melhor até do que o sklearn;
3. A visualização das regiões de decisão são mais claras quando existem grupos bem definidos (clustering).

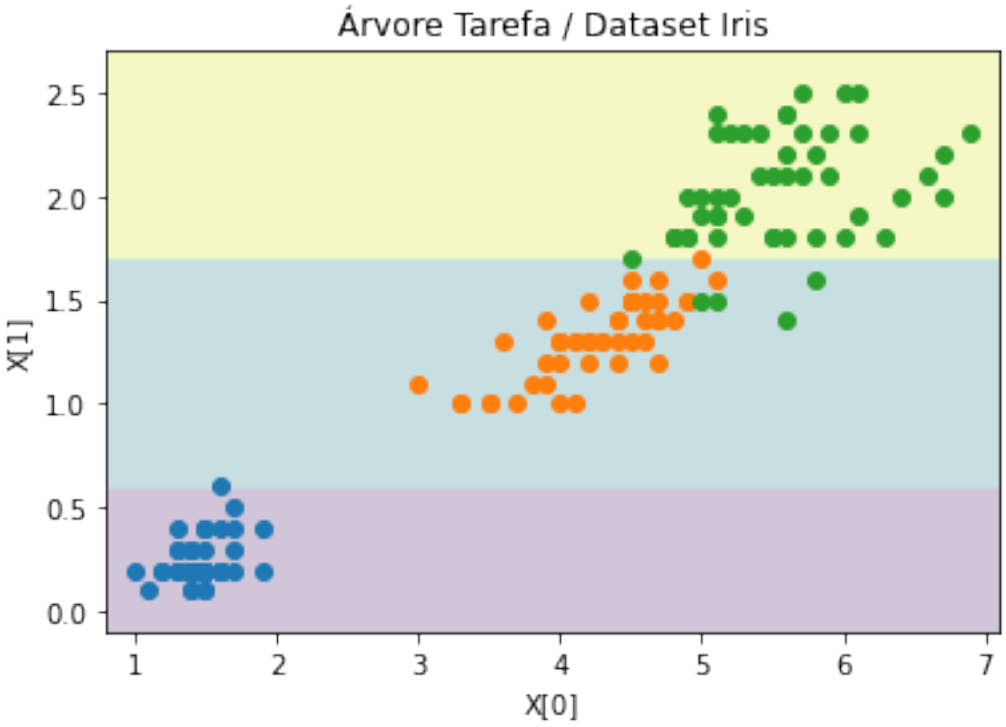
0.8 Todas as regiões de decisão

[64]:
```
from sklearn.tree import DecisionTreeClassifier

print("Dataset Iris")
plotDecisao(DecisionTreeClassifier(), Xi, yi, "Árvore Sklearn", "Dataset Iris")
plotDecisao(ArvoreVideo(), Xi, yi, "Árvore Vídeo", "Dataset Iris")
plotDecisao(ArvoreTarefa(), Xi, yi, "Árvore Tarefa", "Dataset Iris")

print("\nDataset 1")
plotDecisao(DecisionTreeClassifier(), X1, y1, "Árvore Sklearn", "Dataset 1")
plotDecisao(ArvoreVideo(), X1, y1, "Árvore Vídeo", "Dataset 1")
plotDecisao(ArvoreTarefa(), X1, y1, "Árvore Tarefa", "Dataset 1")

print("\nDataset 2")
plotDecisao(DecisionTreeClassifier(), X2, y2, "Árvore Sklearn", "Dataset 2")
plotDecisao(ArvoreVideo(), X2, y2, "Árvore Vídeo", "Dataset 2")
plotDecisao(ArvoreTarefa(), X2, y2, "Árvore Tarefa", "Dataset 2")

print("\nDataset 3")
plotDecisao(DecisionTreeClassifier(), X3, y3, "Árvore Sklearn", "Dataset 3")
plotDecisao(ArvoreVideo(), X3, y3, "Árvore Vídeo", "Dataset 3")
plotDecisao(ArvoreTarefa(), X3, y3, "Árvore Tarefa", "Dataset 3")
```

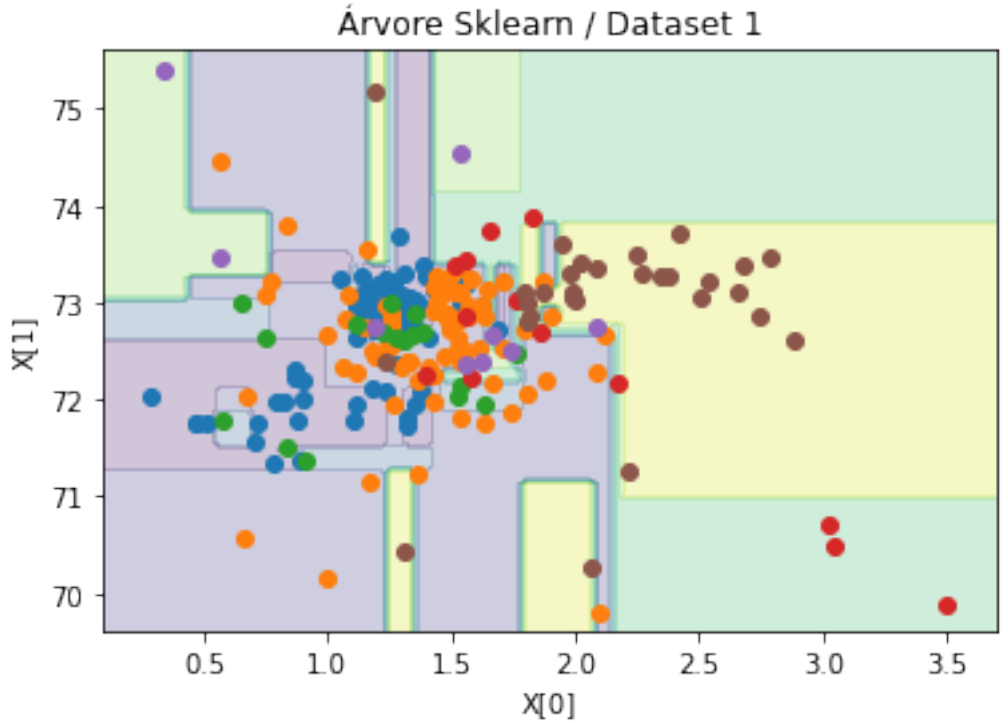
```
Dataset Iris
```

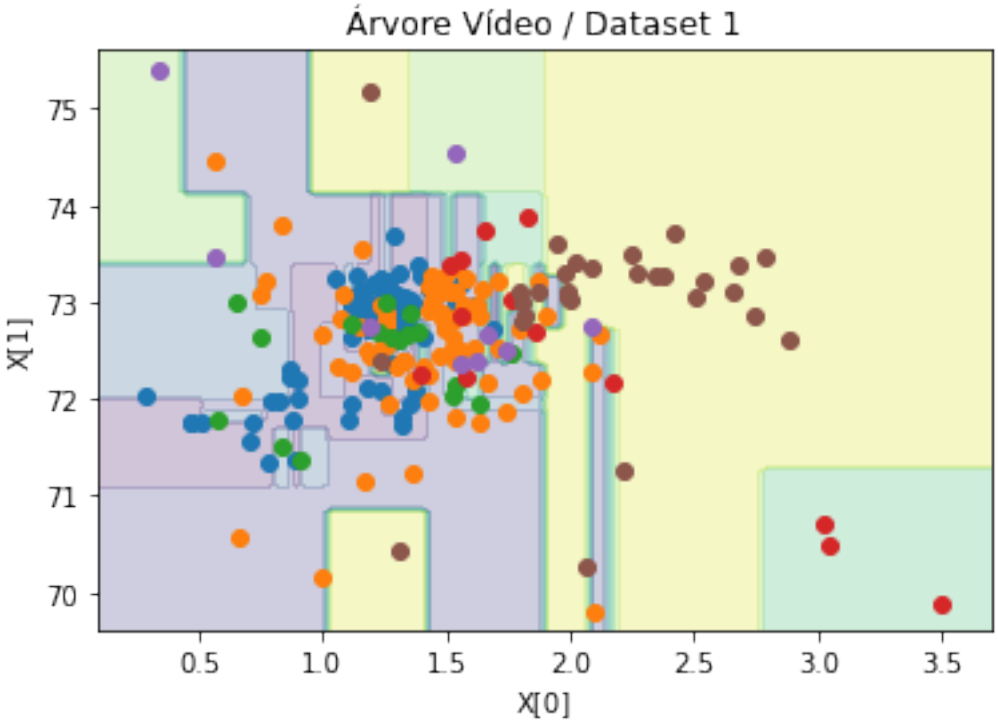

Árvore Sklearn / Dataset Iris
X[1]
X[0]
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

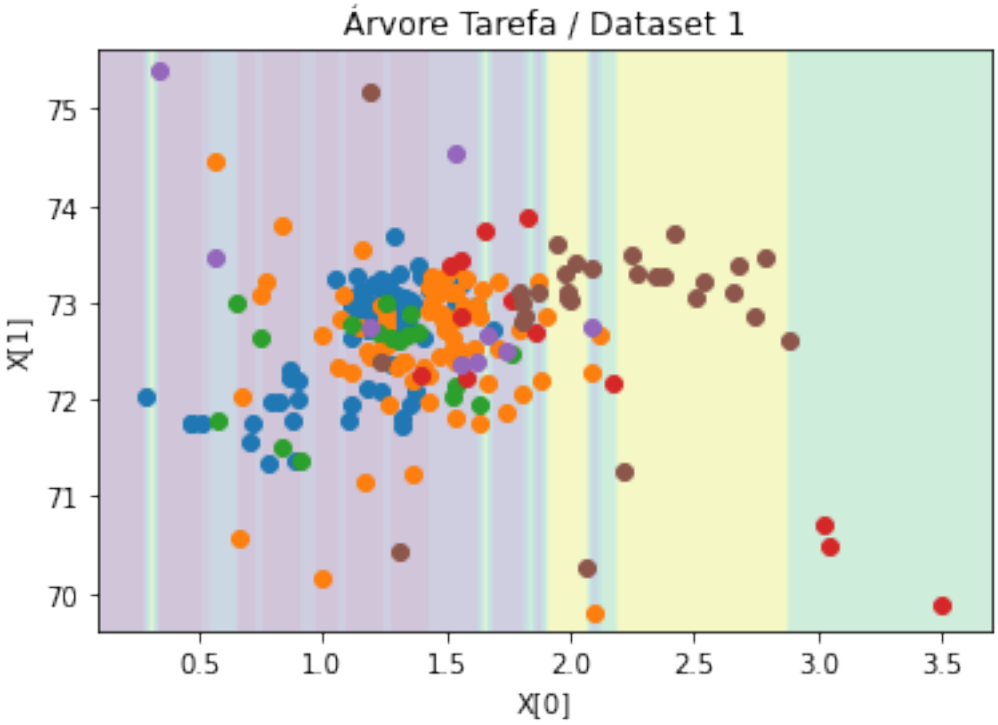

Árvore Vídeo / Dataset Iris
X[1]
X[0]
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7


Árvore Tarefa / Dataset Iris
2.5
2.0
1.5
1.0
0.5
0.0
X[1]
1
2
3
4
5
6
7
X[0]

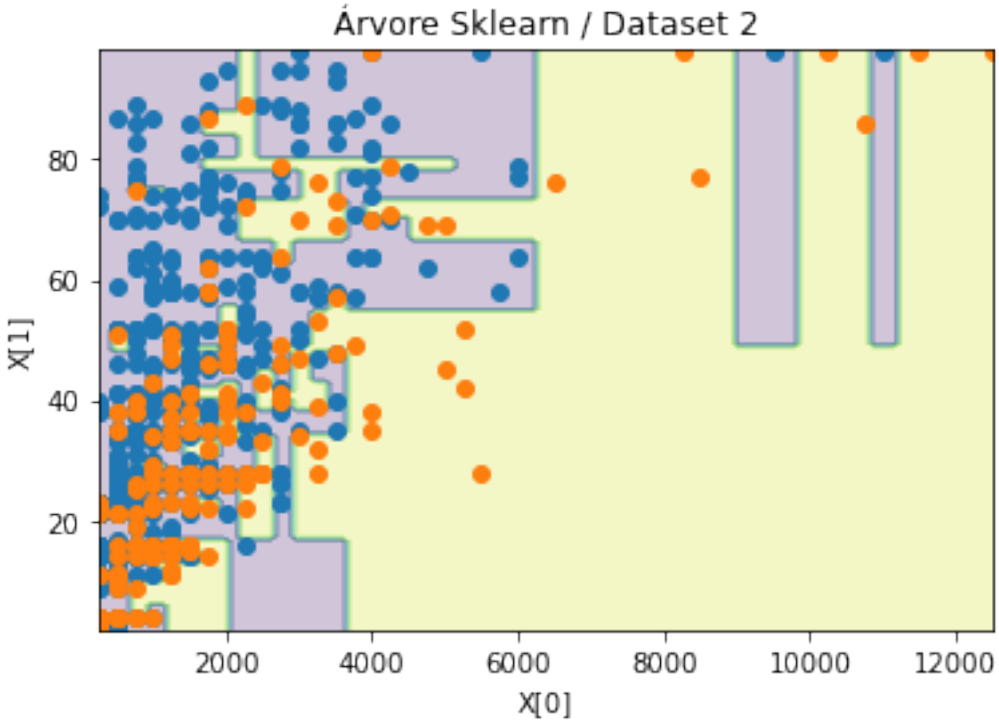
Dataset 1

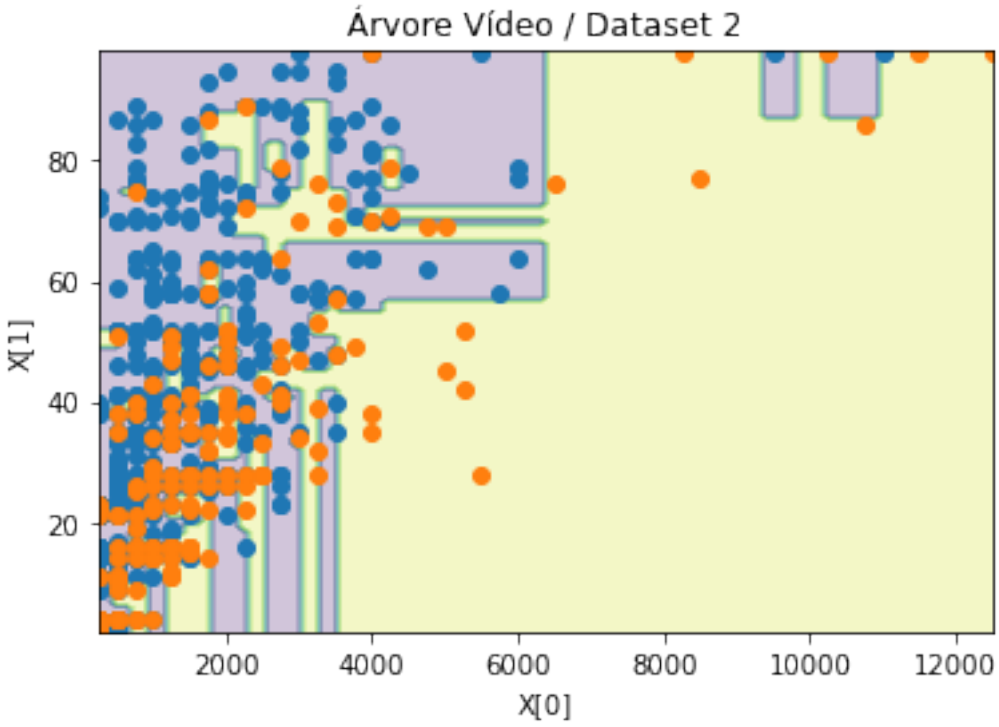

Árvore Sklearn / Dataset 1
75
74
73
72
71
70
X[1]
0.5
1.0
1.5
2.0
2.5
3.0
3.5
X[0]

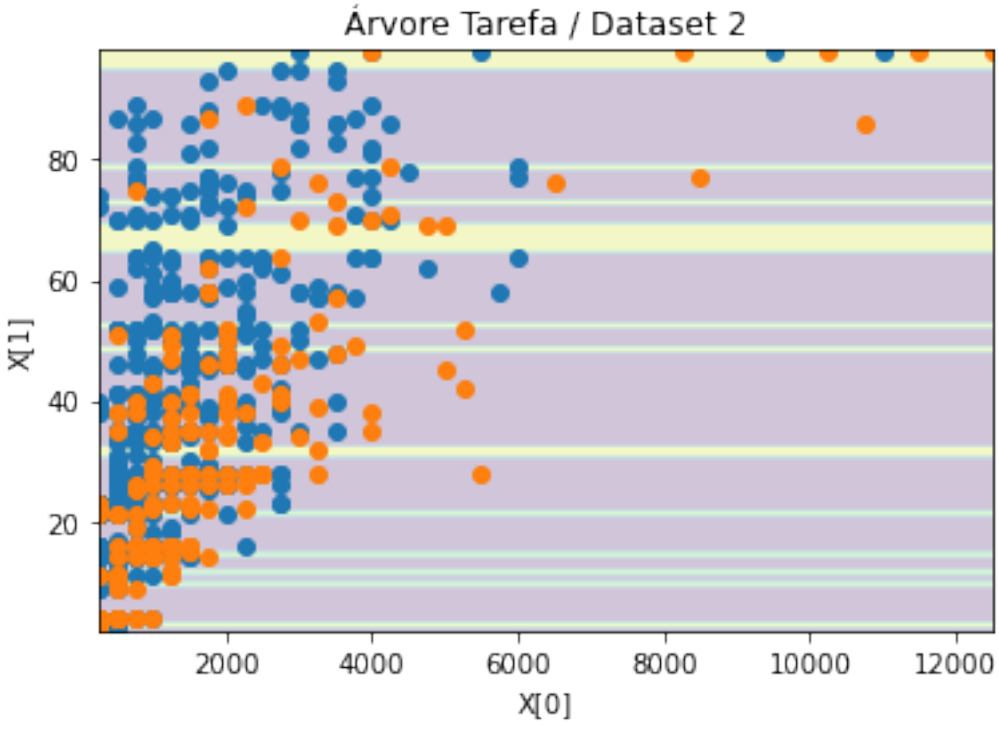

Árvore Vídeo / Dataset 1
75
74
73
72
71
70
X[1]
0.5
1.0
1.5
2.0
2.5
3.0
3.5
X[0]


Árvore Tarefa / Dataset 1
75
74
73
72
71
70
X[1]
0.5
1.0
1.5
2.0
2.5
3.0
3.5
X[0]

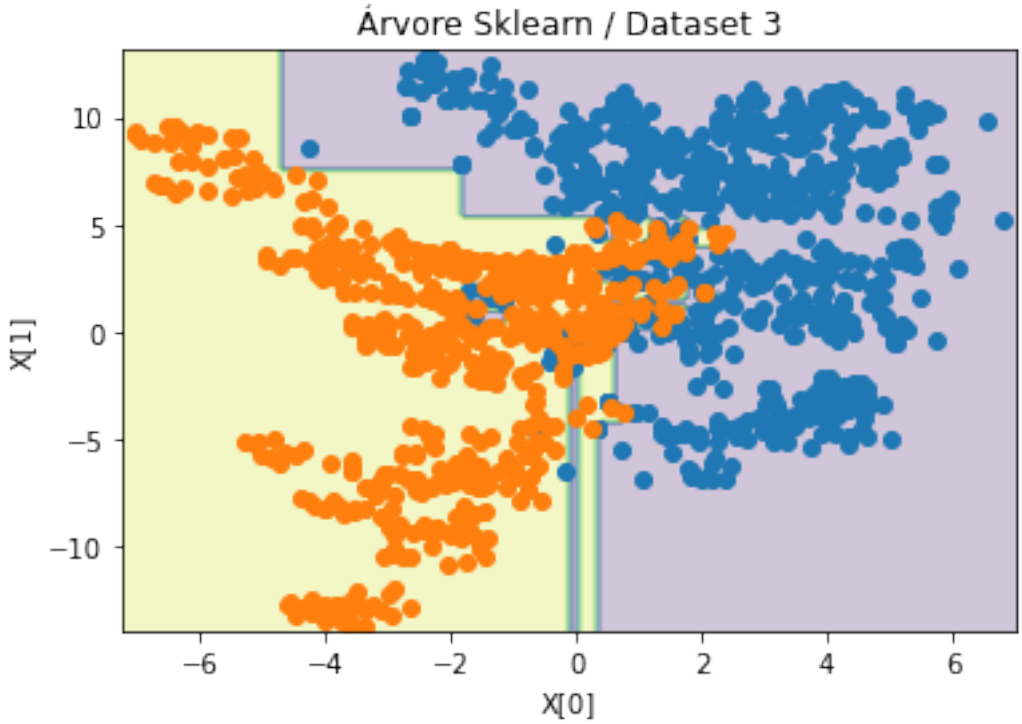
Dataset 2

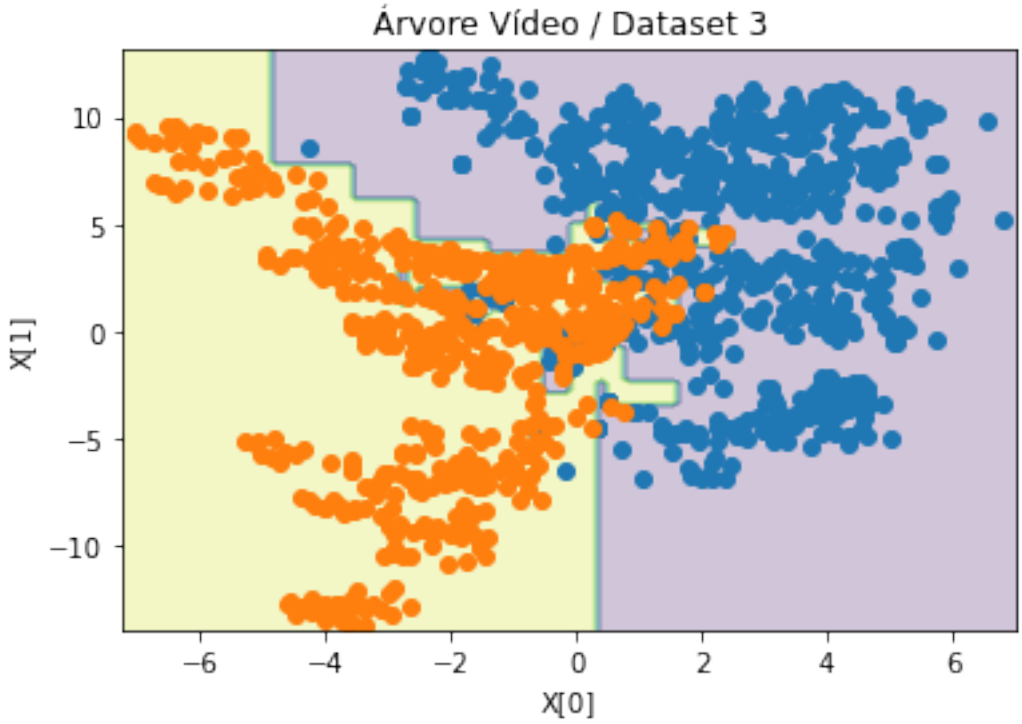

Árvore Sklearn / Dataset 2
80
60
40
20
X[1]
2000
4000
6000
8000
10000
12000
X[0]

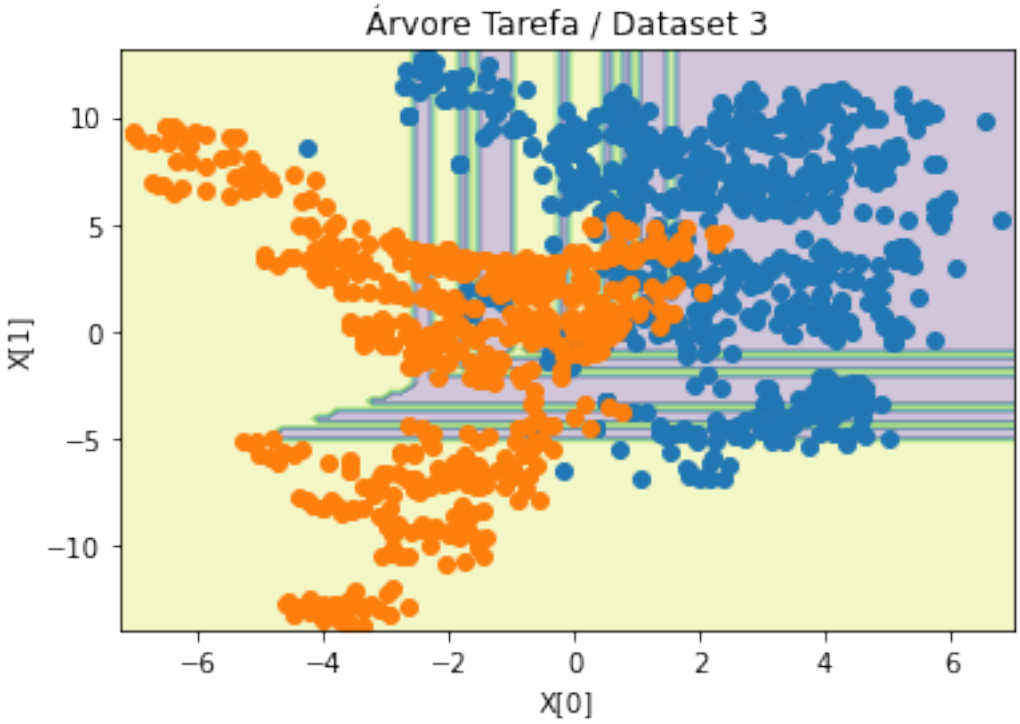

Árvore Vídeo / Dataset 2
80
60
40
20
X[1]
2000
4000
6000
8000
10000
12000
X[0]


Árvore Tarefa / Dataset 2
80
60
40
20
X[1]
2000
4000
6000
8000
10000
12000
X[0]

Dataset 3


Árvore Sklearn / Dataset 3
10
5
0
−5
−10
X[1]
−6
−4
−2
0
2
4
6
X[0]


Árvore Vídeo / Dataset 3
10
5
0
−5
−10
X[1]
−6
−4
−2
0
2
4
6
X[0]


Árvore Tarefa / Dataset 3
10
5
0
−5
−10
X[1]
−6
−4
−2
0
2
4
6
X[0]