In [ ]:
```
import pandas as pd
dataset = pd.read_csv('D:\si\si_lab3_material\dataset\DATASET_SI_XYZ_FINAL.csv', usecols= [0,1,2,3,4,5,6,7,8,9,10,11] )
dataset
```

Out[ ]:

	x_is_at	goto_x	y_is_at	goto_y	z_is_at	goto_z	move_x_right	move_x_left	move_y_outside	move_y_inside	move_z_down	move_z_up
0	5	7	2	2	1	2	1	0	0	0	1	0
1	7	9	2	2	5	5	1	0	0	0	0	0
2	3	10	2	2	1	3	1	0	0	0	1	0
3	3	3	2	2	1	3	0	0	0	0	1	0
4	3	5	2	2	1	4	1	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
19994	1	10	1	3	3	5	0	0	1	0	0	0
19995	10	5	2	3	5	2	0	0	1	0	0	0
19996	5	8	1	2	5	2	0	0	1	0	0	0
19997	3	5	2	3	5	4	0	0	1	0	0	0
19998	10	3	2	3	2	3	0	0	1	0	0	0

19999 rows × 12 columns

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report
from colorama import init, Fore, Back, Style
import numpy as np

X = dataset.iloc[:, 0:6]
Y = dataset.iloc[:20000, 6:12]
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.25, random_state=10, shuffle=False)

clf = DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
clf.get_params()

predictions = clf.predict(X_test)
#predictions = clf.predict_proba(X_test)
#predictions[:10]

#print(y_test)
#print(X_test)
print(X)
```

```
       x_is_at  goto_x  y_is_at  goto_y  z_is_at  goto_z
0            5       7        2       2        1       2
1            7       9        2       2        5       5
2            3      10        2       2        1       3
3            3       3        2       2        1       3
4            3       5        2       2        1       4
...        ...     ...      ...     ...      ...     ...
19994        1      10        1       3        3       5
19995       10       5        2       3        5       2
19996        5       8        1       2        5       2
19997        3       5        2       3        5       4
19998       10       3        2       3        2       3
[19999 rows x 6 columns]
```

In [ ]:
```
#accuracy_score(y_test, predictions)                    # accuracy
#confusion_matrix(y_test, predictions, labels=[0,1])    # [num corretas, num erradas] ou [num erradas, num corretas]
#precision_score(y_test, predictions, average=None)                  # precision
#print(classification_report(y_test, predictions, target_names['move_x_right', 'move_x_left']))
print(y_test[:10])
```

```
       move_x_right  move_x_left  move_y_outside  move_y_inside  move_z_down  \
14999             0            0               1              0            0
15000             0            0               1              0            0
15001             0            0               0              0            0
15002             0            0               1              0            0
15003             0            0               0              0            0
15004             0            0               1              0            0
15005             0            0               0              0            0
15006             0            0               0              1            0
15007             0            0               1              0            0
15008             0            0               1              0            0

       move_z_up
14999          0
15000          0
15001          0
15002          0
15003          0
15004          0
15005          0
15006          0
15007          0
15008          0
```

In [ ]:
```
print(X_test[:10])
```

```
       x_is_at  goto_x  y_is_at  goto_y  z_is_at  goto_z
14999        9       3        1       2        5       1
15000        6       4        2       3        1       4
15001        8       6        3       3        1       5
15002       10       4        1       3        5       2
15003        7       5        3       3        5       4
15004        4       4        1       2        3       2
15005        9      10        1       1        5       4
15006        8       8        2       1        5       3
15007        5       3        2       3        2       1
15008        8       2        1       3        2       4
```

In [ ]:
```
accuracy_score(y_test, predictions)
```

Out[ ]:
```
0.999
```

In [ ]:
```
print(X_train[:2])
```

```
   x_is_at  goto_x  y_is_at  goto_y  z_is_at  goto_z
0        5       7        2       2        1       2
1        7       9        2       2        5       5
```

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from colorama import Fore, Back, Style
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import hamming_loss
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Inicialmente procede-se ao load do dataset para o código, através da função pd.read_csv; O parâmetro usecols é usado para evitar o aparecimento de colunas
# extra que não pertencem ao dataset, e que podem causar erros na modelação dos dados

dataset = pd.read_csv('D:\si\si_lab3_material\dataset\DATASET_SI_XYZ_FINAL.csv', usecols= [0,1,2,3,4,5,6,7,8,9,10,11])
dataset
```

Out[ ]:

	x_in_at	goto_x	y_in_at	goto_y	z_in_at	goto_z	move_x_right	move_x_left	move_y_outside	move_y_inside	move_z_down	move_z_up
0	5	7	2	2	1	2	1	0	0	0	1	0
1	7	9	2	2	5	5	1	0	0	0	0	0
2	3	10	2	2	1	3	1	0	0	0	1	0
3	3	3	2	2	1	3	0	0	0	0	1	0
4	3	5	2	2	1	4	1	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
19994	1	10	1	3	3	5	0	0	1	0	0	0
19995	10	5	2	3	5	2	0	0	1	0	0	0
19996	5	8	1	2	5	2	0	0	1	0	0	0
19997	3	5	2	3	5	4	0	0	1	0	0	0
19998	10	3	2	3	2	3	0	0	1	0	0	0

19999 rows × 12 columns

In [ ]:
```
# Primeiro é feita a divisão do dataset em dois sets, X e Y. X reune todos os inputs do dataset, e Y reune todos os outputs. Após isto, a função
# train_test_split faz novamente uma divisão de X e Y em sub sets. X_train_Decision_Tree e Y_train_Decision_Tree são os sets que serão usados para
# treinar o modelo, X_test_Decision_Tree e Y_test serão os sets usados para testar a eficácia do modelo de previsão após o modelo ser treinado.
# O parâmetro test_size indica a percentagem de dados do dataset original que serão usados para o teste de eficácia do modelo. 0.25 é um rácio
# bom para o problema em mãos, uma vez que o prediction resultante apresenta uma grau elevado de accuracy para vários testes realizado
# (geralmente variando entre 0.895 e 1). O modelo usado aqui foi Decision Tree.

X = dataset.iloc[:20000, 0:6]
Y = dataset.iloc[:20000, 6:12]
X_train_Decision_Tree, X_test_Decision_Tree, Y_train_Decision_Tree, Y_test_Decision_Tree = train_test_split(X,Y,test_size=0.25)

clf_DT = DecisionTreeClassifier()
clf_DT = clf_DT.fit(X_train_Decision_Tree, Y_train_Decision_Tree)

prediction_Decision_Tree = clf_DT.predict(X_test_Decision_Tree)

accuracy_Decision_Tree = accuracy_score(Y_test_Decision_Tree,prediction_Decision_Tree)
accuracy_Decision_Tree
```

Out[ ]:
```
0.9952
```

In [ ]:
```
# O resultado do accuracy revela a semelhança entre o conjunto obtido pelo modelo predict e o conjunto Y_Test, que contém a solução correta para
# o input que predict recebe (X_test_Decision_Tree).

accuracy_score(Y_test_Decision_Tree, prediction_Decision_Tree)
```

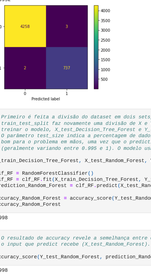
Out[ ]:
```
0.9978
```

In [ ]:
```
# Hamming loss é mais específico no seu resultado de accuracy do que accuracy_score. Enquanto accuracy considera errado uma linha de Output inteira se
# um dos valores na linha estiver errado, Hamming loss apenas considera como errado esse valor em específico para a sua estatística de accuracy,
# em vez de condenar a linha inteira

hamming_loss(Y_test_Decision_Tree, prediction_Decision_Tree)
```

Out[ ]:
```
0.0008
```

In [ ]:
```
# Ao longo deste notebook encontram-se alguns testes com uso de Confusion Matrix, para ambos os modelos e para casos estáticos com recurso a arrays definidos.
# Canto superior Direito e Inferior Esquerdo representa Previsões corretas, Canto superior Esquerdo e Inferior Direito representa Previsões incorretas

acc = accuracy_score(Y_test_Decision_Tree, prediction_Decision_Tree)
mat = confusion_matrix(Y_test_Decision_Tree.iloc[:,0],prediction_Decision_Tree[:,0])
mat_display = ConfusionMatrixDisplay(confusion_matrix=mat)
mat_display.plot()
print(acc)
```

Out[ ]:
```
0.9952
```



In [ ]:
```
# Primeiro é feita a divisão do dataset em dois sets, X e Y. X reune todos os inputs do dataset, e Y reune todos os outputs. Após isto, a função
# train_test_split faz novamente uma divisão de X e Y em sub sets. X_train_Random_Forest e Y_train_Random_Forest são os sets que serão usados para
# treinar o modelo, X_test_Decision_Tree_Forest e Y_test serão os sets usados para testar a eficácia do modelo de previsão após o modelo ser treinado.
# O parâmetro test_size indica a percentagem de dados do dataset original que serão usados para o teste de eficácia do modelo. 0.25 é um rácio
# bom para o problema em mãos, uma vez que o prediction resultante apresenta uma grau elevado de accuracy para vários testes realizado
# (geralmente variando entre 0.895 e 1). O modelo usado aqui foi Decision_tree Forest.

X_train_Decision_Tree_Forest, X_test_Random_Forest, Y_train_Decision_Tree_Forest, Y_test_Random_Forest = train_test_split(X,Y,test_size=0.25)

clf_RF = RandomForestClassifier()
clf_RF = clf_RF.fit(X_train_Decision_Tree_Forest, Y_train_Decision_Tree_Forest)
prediction_Random_Forest = clf_RF.predict(X_test_Random_Forest)

accuracy_Random_Forest = accuracy_score(Y_test_Random_Forest,prediction_Random_Forest)
accuracy_Random_Forest
```

Out[ ]:
```
0.998
```

In [ ]:
```
# O resultado do accuracy revela a semelhança entre o conjunto obtido pelo modelo predict e o conjunto Y_Test, que contém a solução correta para
# o input que predict recebe (X_test_Random_Forest).

accuracy_score(Y_test_Random_Forest, prediction_Random_Forest)
```

Out[ ]:
```
0.998
```

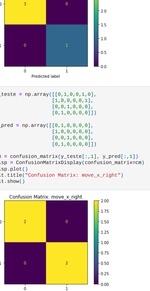
In [ ]:
```
# Hamming loss é mais específico no seu resultado de accuracy do que accuracy_score. Enquanto accuracy considera errado uma linha de Output inteira se
# um dos valores na linha estiver errado, Hamming loss apenas considera como errado esse valor em específico para a sua estatística de accuracy,
# em vez de condenar a linha inteira

hamming_loss(Y_test_Random_Forest, prediction_Random_Forest)
```

Out[ ]:
```
0.0003333333333333333
```

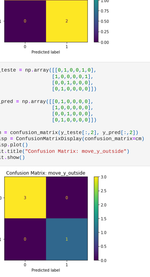
In [ ]:
```
acc = accuracy_score(Y_test_Random_Forest, prediction_Decision_Tree)
mat = confusion_matrix(Y_test_Decision_Tree.iloc[:,0],prediction_Decision_Tree[:,0])
mat_display = ConfusionMatrixDisplay(confusion_matrix=mat)
mat_display.plot()
print(acc)
```

Out[ ]:
```
0.17
```

In [ ]:
```
y_teste = np.array([[0,1,0,0,1,0],
                    [1,0,0,0,0,1],
                    [0,0,1,0,0,0],
                    [0,1,0,0,0,0]])

y_pred = np.array([[0,1,0,0,0,0],
                   [1,0,0,0,0,0],
                   [0,0,1,0,0,0],
                   [0,1,0,0,0,0]])

cm = confusion_matrix(y_teste[:,0], y_pred[:,0])
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix: move_x_left")
plt.show()
```



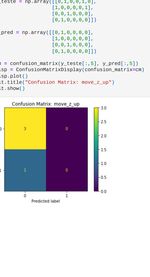
In [ ]:
```
y_teste = np.array([[0,1,0,0,1,0],
                    [1,0,0,0,0,1],
                    [0,0,1,0,0,0],
                    [0,1,0,0,0,0]])

y_pred = np.array([[0,1,0,0,0,0],
                   [1,0,0,0,0,0],
                   [0,0,1,0,0,0],
                   [0,1,0,0,0,0]])

cm = confusion_matrix(y_teste[:,1], y_pred[:,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix: move_x_right")
plt.show()
```



In [ ]:
```
y_teste = np.array([[0,1,0,0,1,0],
                    [1,0,0,0,0,1],
                    [0,0,1,0,0,0],
                    [0,1,0,0,0,0]])

y_pred = np.array([[0,1,0,0,0,0],
                   [1,0,0,0,0,0],
                   [0,0,1,0,0,0],
                   [0,1,0,0,0,0]])

cm = confusion_matrix(y_teste[:,2], y_pred[:,2])
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix: move_y_outside")
plt.show()
```

In [ ]:
```
y_teste = np.array([[0,1,0,0,1,0],
                    [1,0,0,0,0,1],
                    [0,0,1,0,0,0],
                    [0,1,0,0,0,0]])

y_pred = np.array([[0,1,0,0,0,0],
                   [1,0,0,0,0,0],
                   [0,0,1,0,0,0],
                   [0,1,0,0,0,0]])

cm = confusion_matrix(y_teste[:,3], y_pred[:,3])
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix: move_y_inside")
plt.show()
```

In [ ]:
```
y_teste = np.array([[0,1,0,0,1,0],
                    [1,0,0,0,0,1],
                    [0,0,1,0,0,0],
                    [0,1,0,0,0,0]])

y_pred = np.array([[0,1,0,0,0,0],
                   [1,0,0,0,0,0],
                   [0,0,1,0,0,0],
                   [0,1,0,0,0,0]])

cm = confusion_matrix(y_teste[:,4], y_pred[:,4])
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix: move_z_down")
plt.show()
```



In [ ]:
```
y_teste = np.array([[0,1,0,0,1,0],
                    [1,0,0,0,0,1],
                    [0,0,1,0,0,0],
                    [0,1,0,0,0,0]])

y_pred = np.array([[0,1,0,0,0,0],
                   [1,0,0,0,0,0],
                   [0,0,1,0,0,0],
                   [0,1,0,0,0,0]])

cm = confusion_matrix(y_teste[:,5], y_pred[:,5])
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix: move_z_up")
plt.show()
```