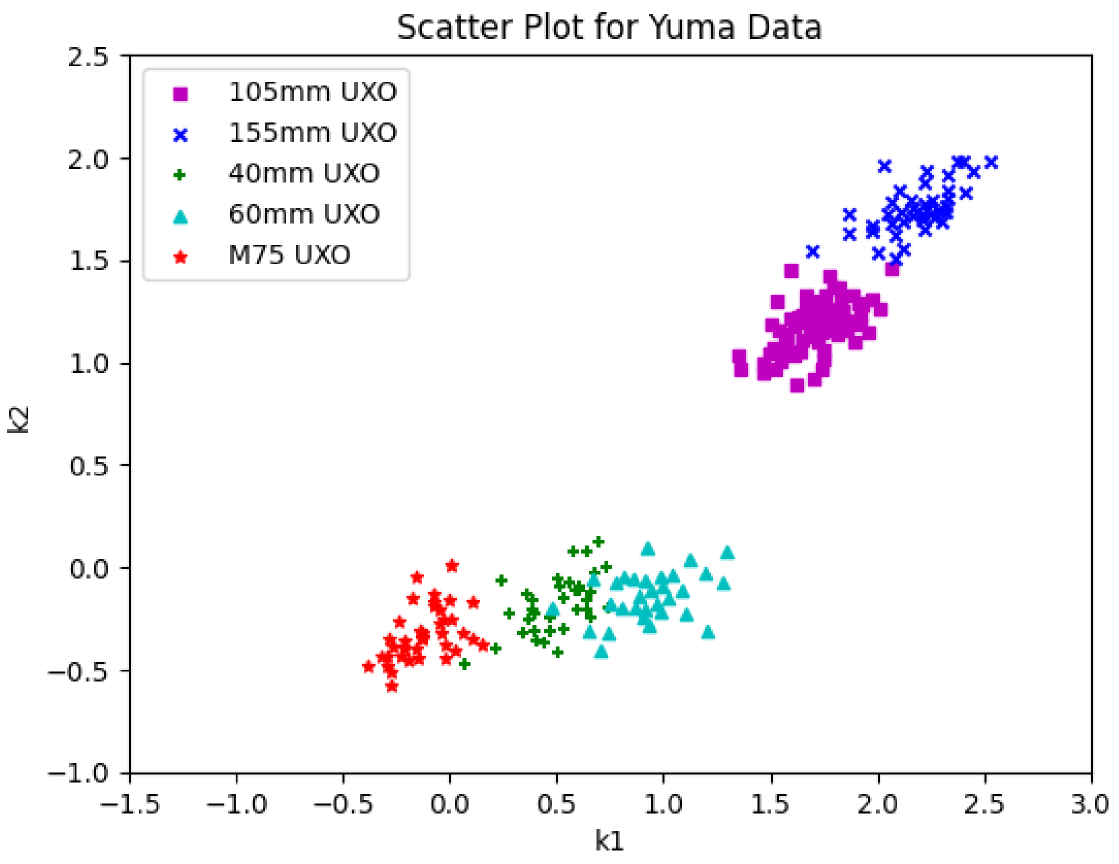
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from A6_functions import measurePerformance, groupScatter, plotGNB,
plotTree

xlim = [-1.5,3]
ylim = [-1,2.5]
colors  = {'155mm':'b', '105mm':'m', 'M75':'r', '60mm':'c',
'40mm':'g'}
markers = {'155mm':'x', '105mm':'s', 'M75':'*', '60mm':'^',
'40mm':'+'}

xlabel = 'k1'
ylabel = 'k2'
markersize = 28
linewidth = 0
random_state = 1

uxo_all = pd.read_csv('Yuma_all.csv')
groups_all = uxo_all.groupby('item')
print(groups_all)
x_all = uxo_all[['k1','k2']]  #features
y_all = uxo_all['item'] #labels
x_all = x_all.values
x_all0 = x_all
y_all0 = y_all
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at
0x7be339934460>
```

```python
groupScatter(groups_all,colors,markers,xlim,ylim,"k1","k2",20,-1,"
UXO")
plt.title("Scatter Plot for Yuma Data")
plt.legend(loc='upper left')
```

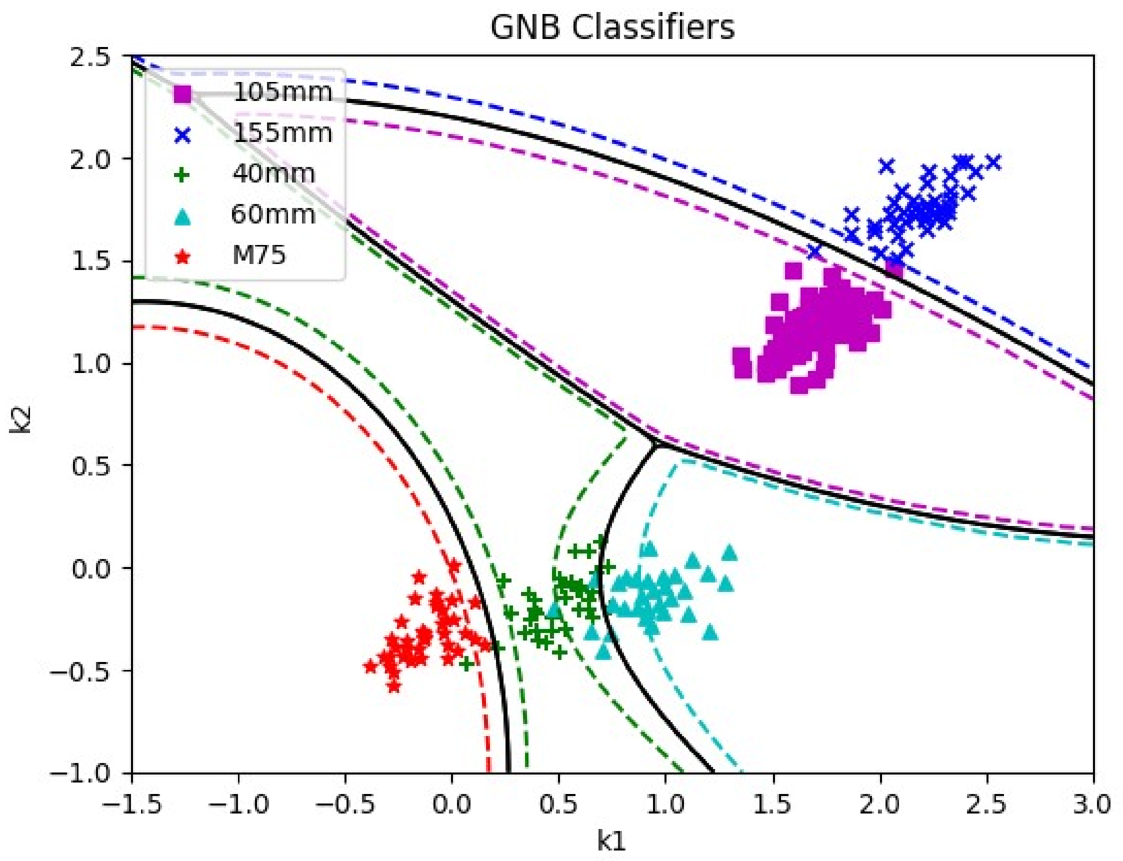
```
/usr/local/lib/python3.10/dist-packages/pandas/plotting/_matplotlib/
core.py:1259: UserWarning: No data for colormapping provided via 'c'.
Parameters 'cmap' will be ignored
  scatter = ax.scatter(
```

```
<matplotlib.legend.Legend at 0x7be339bf62f0>
```

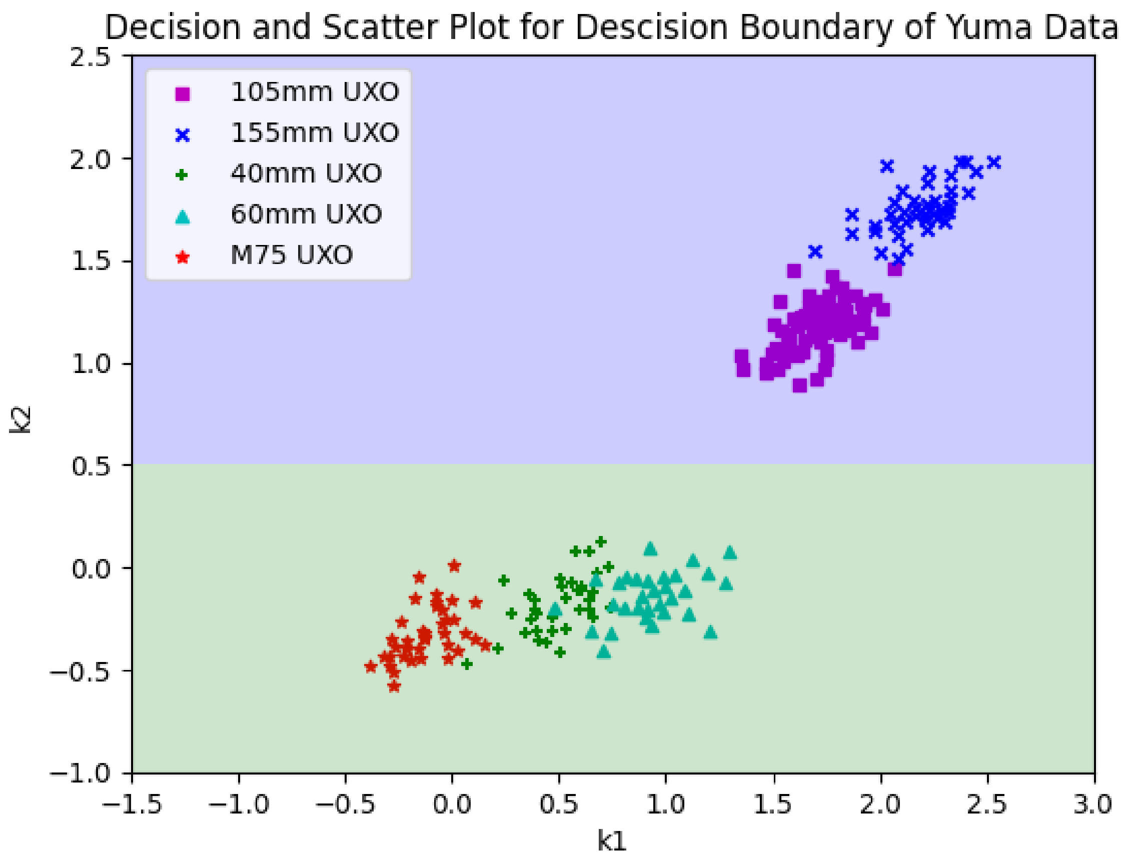


```
model = GaussianNB()
model.fit(x_all,y_all)
plotGNB(model,groups_all,colors,xlim,ylim)
groupScatter(groups_all,colors,markers,xlim,ylim,xlabel,ylabel,markers
ize,linewidth,'')
plt.title('GNB Classifiers')
```

```
/usr/local/lib/python3.10/dist-packages/pandas/plotting/_matplotlib/
core.py:1259: UserWarning: No data for colormapping provided via 'c'.
Parameters 'cmap' will be ignored
  scatter = ax.scatter(

Text(0.5, 1.0, 'GNB Classifiers')
```



```
groupScatter(groups_all,colors,markers,xlim,ylim,"k1","k2",20,-1,"
UXO")
model = DecisionTreeClassifier(max_depth=1,random_state=5)
model.fit(x_all,y_all)
plotTree(model,groups_all,colors,xlim,ylim)
plt.title("Decision and Scatter Plot for Descision Boundary of Yuma
Data")
plt.legend(loc='upper left')
plt.xlabel("k1")
plt.ylabel("k2")

Text(0, 0.5, 'k2')
```

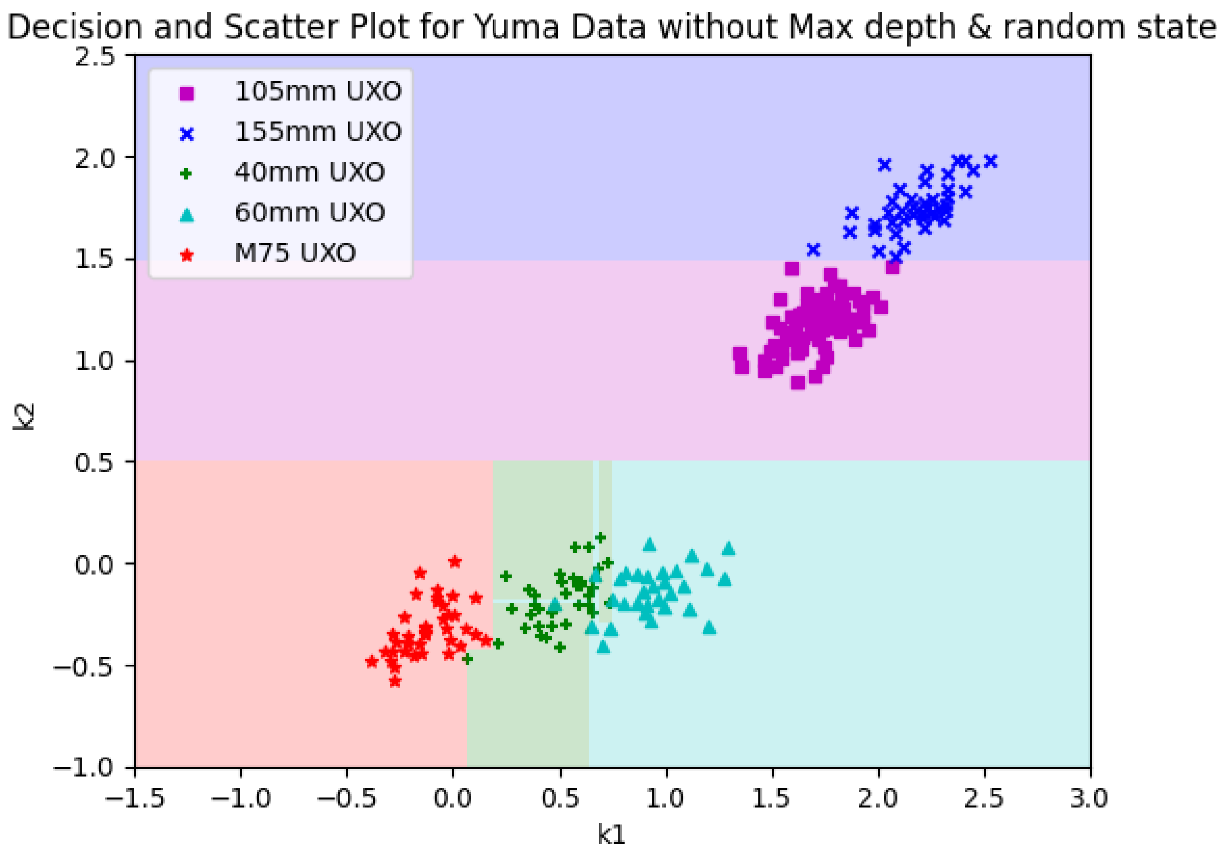


1)In the decision tree plots, each level represents a step where the algorithm divides the data into regions based on specific features to improve classification. At the first level, the dataset is split into two major regions, and subsequent levels further divide these regions into smaller subregions. Not all regions are split at each level, and the decision is based on the information gain or purity improvement achieved by the split. Continuing to more levels would result in finer divisions, but there is a point where further splits may not significantly enhance classification and could lead to overfitting, capturing noise instead of meaningful patterns. The decision tree adapts its structure to the data distribution, creating decision boundaries that best separate classes at each iteration.

```
groupScatter(groups_all,colors,markers,xlim,ylim,"k1","k2",20,-1,"
UXO")
model = DecisionTreeClassifier(max_depth=None,random_state=None)
model.fit(x_all,y_all)
plotTree(model,groups_all,colors,xlim,ylim)
plt.title("Decision and Scatter Plot for Yuma Data without Max depth &
random state")
plt.legend(loc='upper left')
plt.xlabel("k1")
plt.ylabel("k2")
```

```
/usr/local/lib/python3.10/dist-packages/pandas/plotting/_matplotlib/
core.py:1259: UserWarning: No data for colormapping provided via 'c'.
```

```
Parameters 'cmap' will be ignored
  scatter = ax.scatter(

Text(0, 0.5, 'k2')
```



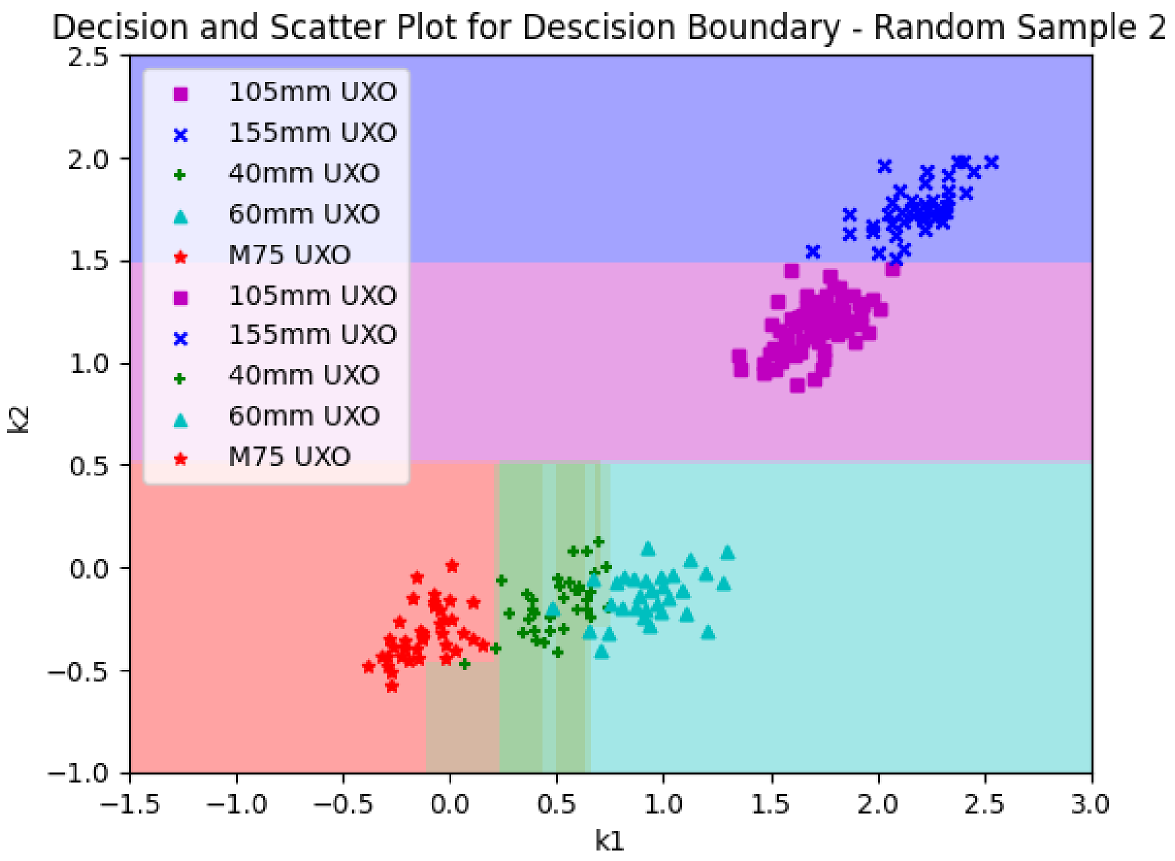
2)When the decision tree is set free without specific rules on depth or decision-making, it dives deep into the data, striving to perfectly fit every detail. This can lead to overfitting, where it gets too fixated on small nuances and even noise in the dataset. Running the process multiple times results in different plots due to the tree's random choices during each run. The variability in plots illustrates how the decision tree, without constraints, adapts excessively to intricate data details, highlighting the risk of overfitting.

```
# Splitting the Data for the First Random Sample
train1, test1 = train_test_split(uxo_all, test_size=0.5,
random_state=1)
train_f1 = train1[['k1', 'k2']].values
train_l1 = train1['item']
test_f1 = test1[['k1', 'k2']].values
test_l1 = test1['item']

# Splitting the Data for the Second Random Sample
train2, test2 = train_test_split(uxo_all, test_size=0.5,
random_state=2)
```

```
train_f2 = train2[['k1', 'k2']].values
train_l2 = train2['item']
test_f2 = test2[['k1', 'k2']].values
test_l2 = test2['item']

# Create a Decision Tree Classifier
model = DecisionTreeClassifier()

# Fit and Plot for Random Sample 1
model.fit(train_f1, train_l1)
plotTree(model, groups_all, colors, xlim, ylim)
groupScatter(groups_all,colors,markers,xlim,ylim,"k1","k2",20,-1,"
UXO")
plt.title("Decision and Scatter Plot for Descision Boundary - Random
Sample 1")
plt.xlabel("k1")
plt.ylabel("k2")
plt.legend(loc='upper left')

# Fit and Plot for Random Sample 2
model.fit(train_f2, train_l2)
plotTree(model, groups_all, colors, xlim, ylim)
groupScatter(groups_all,colors,markers,xlim,ylim,"k1","k2",20,-1,"
UXO")
plt.title("Decision and Scatter Plot for Descision Boundary - Random
Sample 2")
plt.xlabel("k1")
plt.ylabel("k2")
plt.legend(loc='upper left')
```

```
/usr/local/lib/python3.10/dist-packages/pandas/plotting/_matplotlib/
core.py:1259: UserWarning: No data for colormapping provided via 'c'.
Parameters 'cmap' will be ignored
  scatter = ax.scatter(
/usr/local/lib/python3.10/dist-packages/pandas/plotting/_matplotlib/
core.py:1259: UserWarning: No data for colormapping provided via 'c'.
Parameters 'cmap' will be ignored
  scatter = ax.scatter(
```

```
<matplotlib.legend.Legend at 0x7be338bb2050>
```

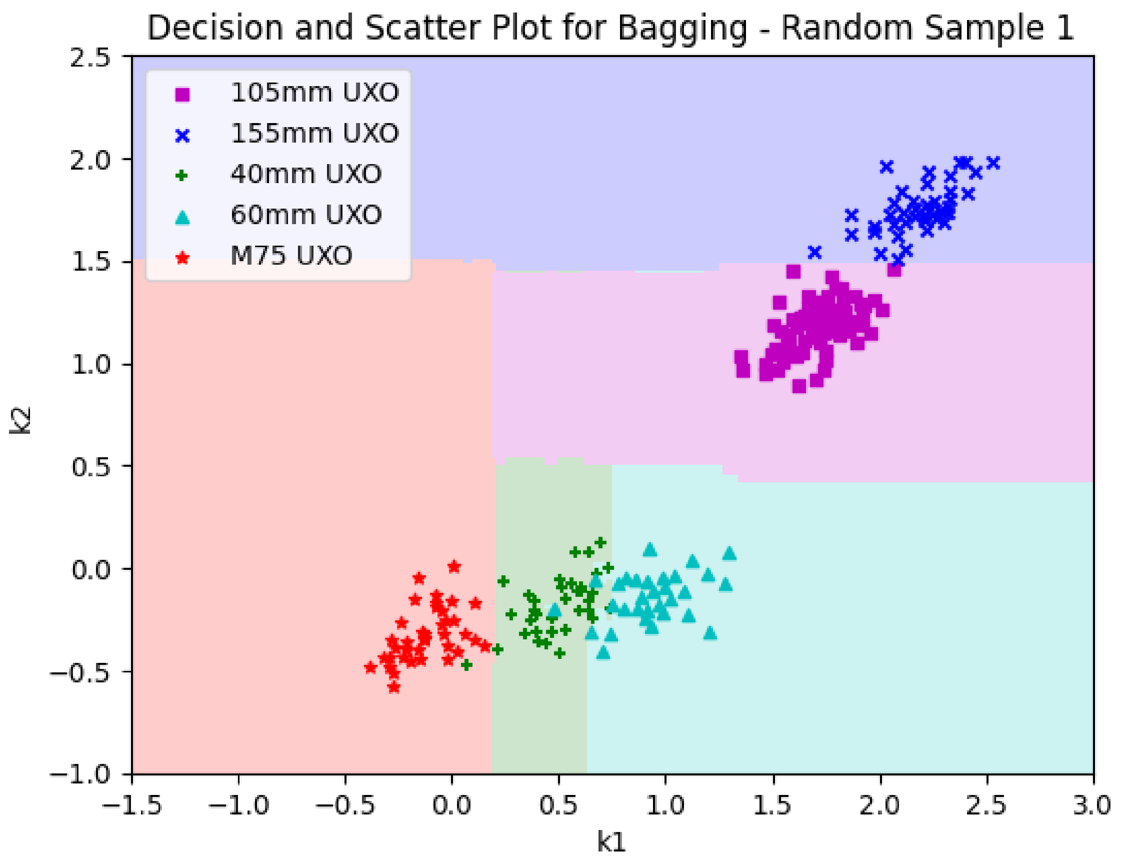


3)The decision boundaries in the plots differ due to the randomness introduced during the train-test split. Each run of the Decision Tree Classifier is performed on a unique random sample, leading to varied decision boundaries. The distinct plots highlight the model's sensitivity to different subsets of the data, emphasizing the impact of random sampling on the learning process.

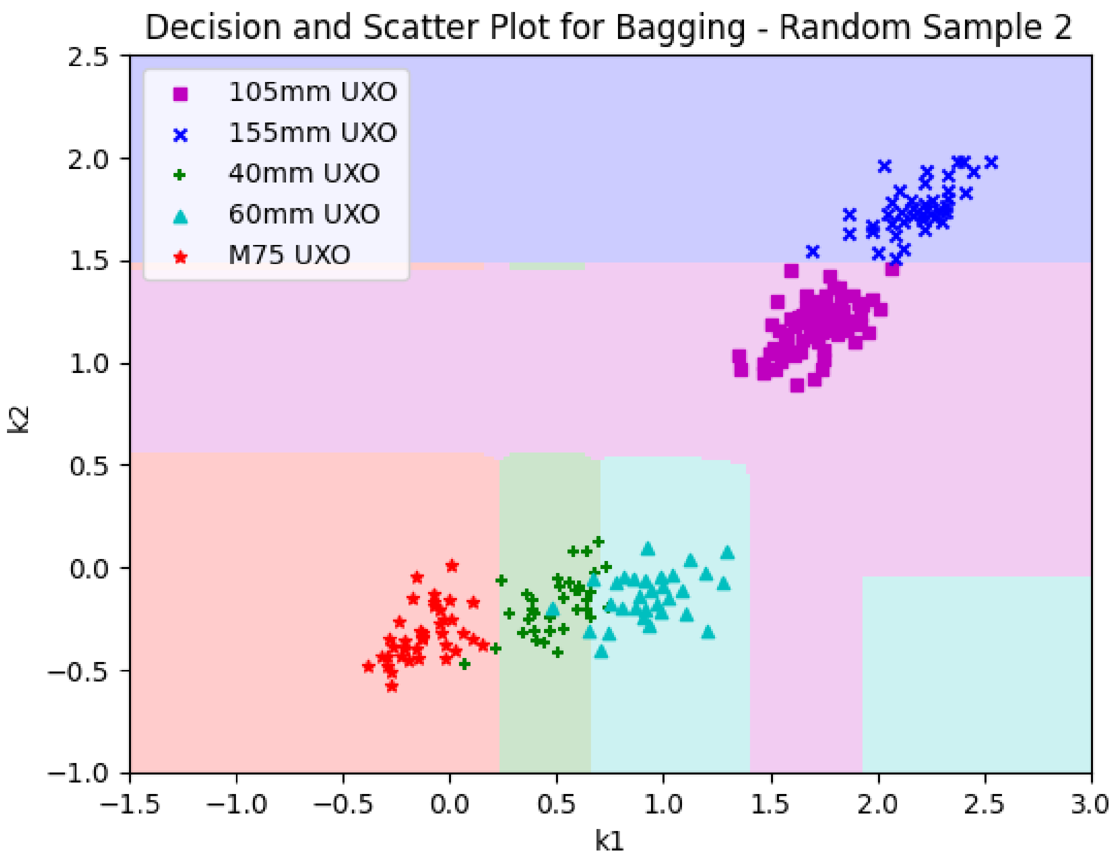
```
# Create a Decision Tree Classifier
tree = DecisionTreeClassifier()

# Fit and Plot for Bagging on Random Sample 1
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8,
random_state=1)
bag.fit(train_f1, train_l1)
plotTree(bag, groups_all, colors, xlim, ylim)
groupScatter(groups_all, colors, markers, xlim, ylim, "k1", "k2", 20,
-1, " UXO")
plt.title("Decision and Scatter Plot for Bagging - Random Sample 1")
plt.xlabel("k1")
plt.ylabel("k2")
plt.legend(loc='upper left')
plt.show()

# Fit and Plot for Bagging on Random Sample 2
```

```
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8,
random_state=2)
bag.fit(train_f2, train_l2)
plotTree(bag, groups_all, colors, xlim, ylim)
groupScatter(groups_all, colors, markers, xlim, ylim, "k1", "k2", 20,
-1, " UXO")
plt.title("Decision and Scatter Plot for Bagging - Random Sample 2")
plt.xlabel("k1")
plt.ylabel("k2")
plt.legend(loc='upper left')

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/pandas/plotting/_matplotlib/
core.py:1259: UserWarning: No data for colormapping provided via 'c'.
Parameters 'cmap' will be ignored
  scatter = ax.scatter(
```



```
/usr/local/lib/python3.10/dist-packages/pandas/plotting/_matplotlib/
core.py:1259: UserWarning: No data for colormapping provided via 'c'.
Parameters 'cmap' will be ignored
  scatter = ax.scatter(
```

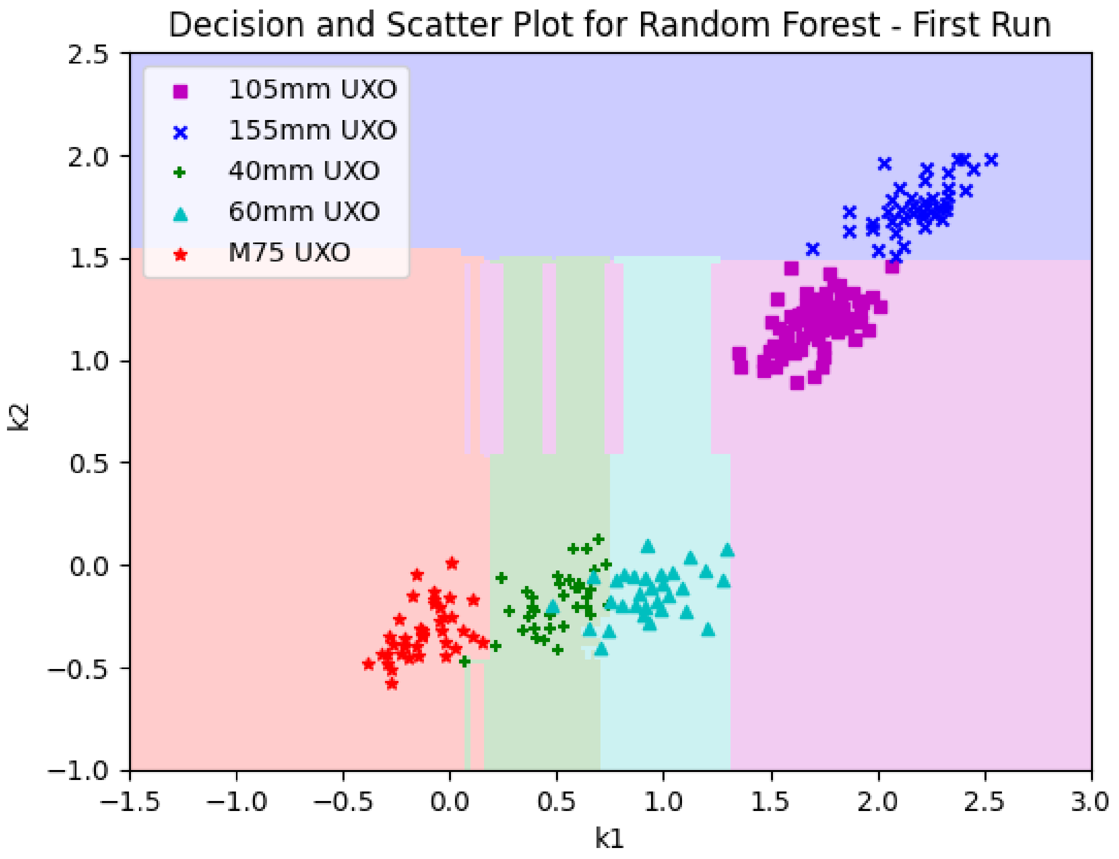


4)For a random forest, key hyperparameters are n_estimators (number of trees) and max_depth (tree depth). Adjusting n_estimators impacts model complexity, while tuning max_depth balances capturing patterns and avoiding overfitting. (Not really sure if this is what the quesion is asking)

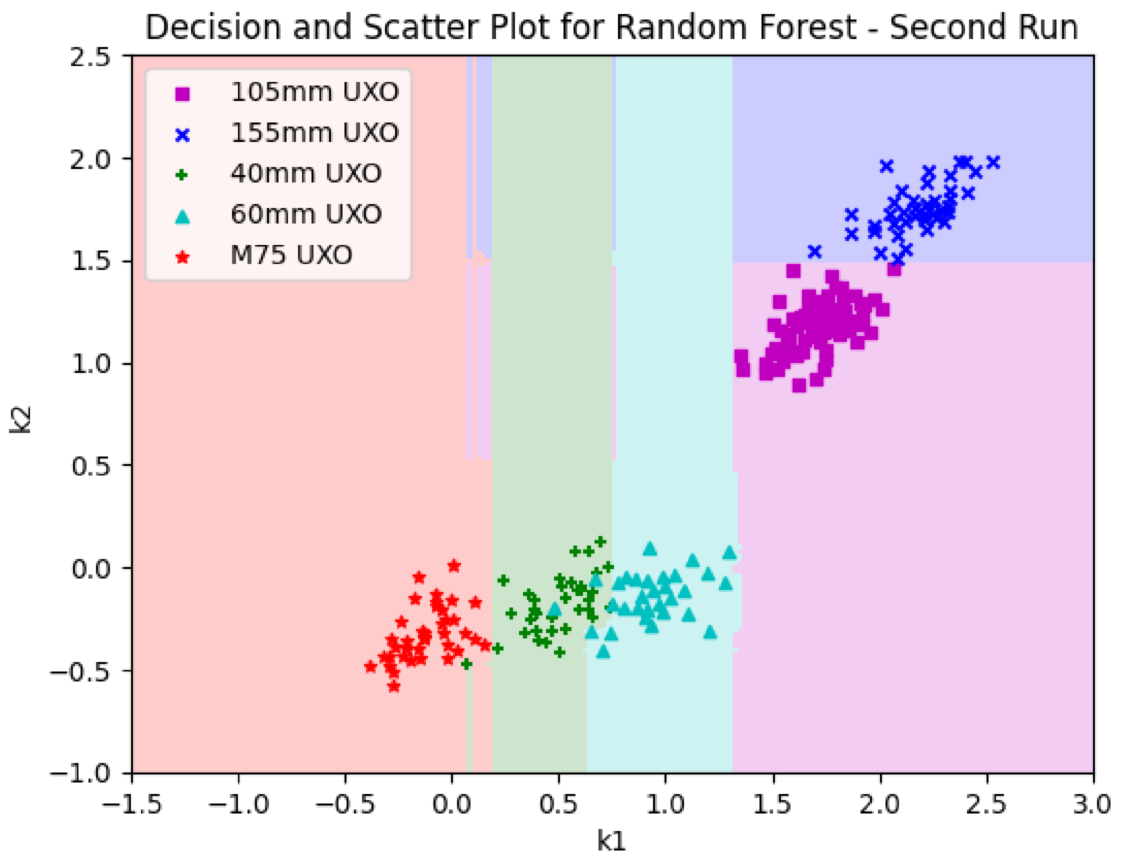
```
# Create a Random Forest Classifier
forest = RandomForestClassifier(n_estimators=100)  # The model is now
named 'forest'.

# Fit and Plot for Random Forest - First Run
forest.fit(x_all, y_all)
plotTree(forest, groups_all, colors, xlim, ylim)
groupScatter(groups_all, colors, markers, xlim, ylim, "k1", "k2", 20,
-1, " UXO")
plt.title("Decision and Scatter Plot for Random Forest - First Run")
plt.xlabel("k1")
plt.ylabel("k2")
plt.legend(loc='upper left')
plt.show()

# Fit and Plot for Random Forest - Second Run
forest.fit(x_all, y_all)
plotTree(forest, groups_all, colors, xlim, ylim)
```

```
groupScatter(groups_all, colors, markers, xlim, ylim, "k1", "k2", 20,
-1, " UXO")
plt.title("Decision and Scatter Plot for Random Forest - Second Run")
plt.xlabel("k1")
plt.ylabel("k2")
plt.legend(loc='upper left')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/pandas/plotting/_matplotlib/
core.py:1259: UserWarning: No data for colormapping provided via 'c'.
Parameters 'cmap' will be ignored
  scatter = ax.scatter(
```



```
/usr/local/lib/python3.10/dist-packages/pandas/plotting/_matplotlib/
core.py:1259: UserWarning: No data for colormapping provided via 'c'.
Parameters 'cmap' will be ignored
  scatter = ax.scatter(
```



5)The Random Forest plots show less variation than the tree classifier plots because Random Forests combine predictions from multiple trees, reducing individual tree fluctuations. The remaining variability comes from diverse trees in the ensemble, aiding generalization without overfitting.

6)The new Random Forest classification models are likely closer to intuition compared to the original simpler decision tree classifiers. Random Forests aggregate predictions from multiple trees, reducing overfitting and capturing more robust patterns in the data. This ensemble approach tends to provide a more balanced and accurate representation of the underlying structure in the parameter spaces, aligning better with real-world expectations.

```
# Load training and test data
train_data = pd.read_csv('Yuma_train2.csv')
test_data = pd.read_csv('Yuma_new2.csv')

# Separate features and labels
X_train = train_data[['k1', 'k2']]
y_train = train_data['item']

X_test = test_data[['k1', 'k2']]
y_test = test_data['item']

# Train Decision Tree Classifier
```

```
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)

# Predict and evaluate Decision Tree performance
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy)

# Train Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100)
rf_classifier.fit(X_train, y_train)

# Predict and evaluate Random Forest performance
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy)
```

```
Decision Tree Accuracy: 1.0
Random Forest Accuracy: 0.9545454545454546
```

7)The Decision Tree and Random Forest classifiers both have an accuracy of 95.45%. They perform similarly in this case. This could be because the dataset is not very complex, and the additional complexity of the Random Forest doesn't provide a significant improvement over a single Decision Tree. In situations where the data is straightforward, a single Decision Tree may perform just as well as a Random Forest.