This lab uses the iris_Data dataset https://www.kaggle.com/datasets/arshid/iris-flower-dataset.

In Activity 2, you are going to explore supervised learning algorithms such as DECISION TREE CLASSIFIER and Random Forest.

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#iris Dataset
df = pd.read_csv("iris_Data.csv")
```

In [ ]:

```
#Printing the first 1o rows of the Dataset
```

In [ ]:

```
df.head(10)
```

In [ ]:

```
# Display the number of Rows and Columns
```

In [ ]:

```
df.shape
```

In [ ]:

```
#Generating the Statistical Measures of the data
```

In [ ]:

```
df.describe()
```

In [ ]:

```
df.dtypes
```

In [ ]:

```
features = list(df.columns[:4])
x=pd.DataFrame(df[['sepal.length', 'sepal.width','petal.length','petal.width']])
y=df['variety']
```

Derive the Train_Test_Split model to analyse the IRIS Dataset

In [ ]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.3, random_state=0)
```

In [ ]:

```
x_train.shape,y_train.shape
```

In [ ]:

```
x_test.shape,y_test.shape
```

In [ ]:

```
#From https://www.analyticsvidhya.com/blog/2021/03/everything-you-need-to-know-about-machine-learning/
from matplotlib import pyplot as plt
from matplotlib import image as mpimg

plt.title("MACHINE LEARNING ALGORITHMS")
plt.xlabel("X pixel scaling")
plt.ylabel("Y pixels scaling")

image = mpimg.imread("ML_Algorithms.jpg")
plt.imshow(image)
plt.show()
```

DECISION TREE CLASSIFIER

In [ ]:

```
#Building the Model
from sklearn import tree
classifier = tree.DecisionTreeClassifier()
classifier.fit(x_train,y_train)
```

Predicting the values

In [ ]:

```
y_pred=classifier.predict(x_test)
```

In [ ]:

```
#Model Evaluation: create classification report and confusion matrix
cn=['setosa', 'versicolor', 'virginica']
from sklearn.metrics import classification_report
y_true = y_test
y_pred =classifier.predict(x_test)

print(classification_report(y_true, y_pred, target_names=cn))
```

There are four ways to check if the predictions are right or wrong:

TN / True Negative: the case was negative and predicted negative TP / True Positive: the case was positive and predicted positive FN / False Negative: the case was positive but predicted negative FP / False Positive: the case was negative but predicted positive

Precision:- Accuracy of positive predictions. Precision = TP/(TP + FP)

Recall:- Fraction of positives that were correctly identified. Recall = TP/(TP+FN)

F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0 F1 Score = 2* (Recall * Precision) / (Recall + Precision)

Support is the number of actual occurrences of the class in the specified dataset

confusion_matrix

In [ ]:

```
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test,y_pred)

cf
```

In [ ]:

```
import seaborn as sn

sn.heatmap(cf, annot= True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

In [ ]:

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

In [ ]:

```
classifier.predict([[6,3.05,3,1.19]])
```

RANDOM FOREST CLASSIFIER

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
#n_estimators define the underlining decision tree in a Random Forest
clf = RandomForestClassifier(n_estimators=10)
#Train the model usingthe trainingsets y_predict= clf.predict(x_test)
clf.fit(x_train, y_train)

#Prediction on test set
y_pred=clf.predict(x_test)
```

In [ ]:

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

In [ ]:

```
from sklearn.metrics import accuracy_score
#Model Accuracy
print("Accuracy =", accuracy_score(y_test, y_pred))
```

In [ ]:

```
clf.predict([[6,3.05,3,1.19]])
```

In [ ]:

```
#%pip install pydotplus
%pip install graphviz
```

from IPython.display import Image from six import StringIO import pydotplus

dot_data = StringIO() tree.export_graphviz(classifier, out_file=dot_data, feature_names= features) graph = pydotplus.graph_from_dot_data(dot_data.getvalue()) Image(graph.create_png())

In [ ]: