Assignment 1

Student ID: 21235117
Name: Diksha Srivastava
Course: Data Analytics(CSD1)

1. Several Machine Learning Open-Source packages are available like:
   - **Weka**, which is based on java and is a GUI workbench.
   - **Scikit-Learn** is based on Python.
   - **Spark MLlib** based on java, Tensor Flow, and many more.

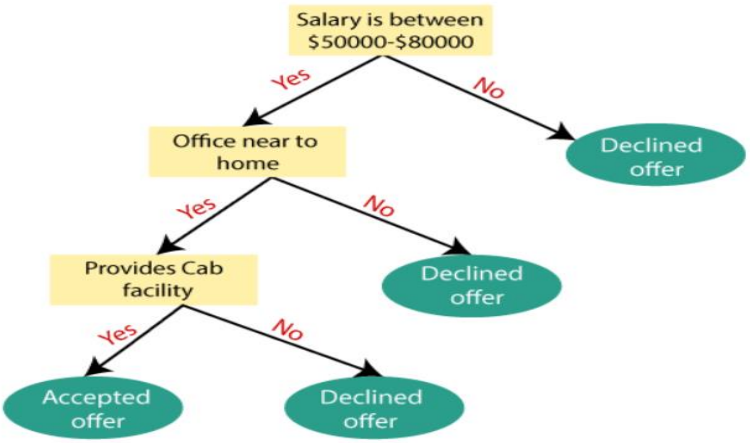
   For the given task, I chose **Scikit-Learn** as:
   - Its documentation is easy to understand, and it is easy to implement as well especially for beginners.
   - It also provides API documentation; hence, it can be easily integrated with other platforms.
   - It provides us with most of the ML models like classification, regression, clustering, and many more.

   Features of Scikit Learn:
   - It already comes with several built-in datasets like iris (for flowers), house prices, diabetes, and many more.
   - It already has packages for data pre-processing, dividing training and test set, feature scaling, Regression model, Decision Trees, KNN model, and many more.
   - It also provides functionality to visualize datasets, for example, creating decision trees, etc.

2. Since Scikit-Learn requires DataFrame as input, the following steps were performed for pre-processing:
   - First, the txt file was converted to a CSV file by importing the txt in excel, delimited by space, and then save in CSV format.
   - Second, the data was loaded into a DataFrame using **Pandas**.
   - The data was split into x_train, y_train, x_test, y_test by fetching the Fire column as y_train which is a dependent variable, and other features in x_train which are independent variables.
   - Since the data in the Fire column had extra spaces, **strip**() was used to remove these spaces from y_train and y_test.

```
train = pd.read_csv('TrainingSet.csv')
test = pd.read_csv('TestSet.csv')
x_train = train[["Year", "Temp", "Humidity", "Rainfall", "Drought_Code", "Buildup_index", "Day", "Month", "Wind_Speed"]]
y_train = train["Fire"].str.strip()
x_test = test[["Year", "Temp", "Humidity", "Rainfall", "Drought_Code", "Buildup_index", "Day", "Month", "Wind_Speed"]]
y_test = np.asarray(test["Fire"].str.strip())
```

3. **ID3:**
   - ID3 algorithm generates a **Decision Tree** for the given dataset. The decision tree is a **supervised learning technique** i.e., works on labeled data. It is also called as **eager learning** approach as it constructs a model for future prediction and deals better with noise.
   - To predict the class, a tree is generated having root node as the feature which has maximum effect on the outcome. Once the root node is selected, a decision is taken on feature values and then a subtree is generated which works recursively. Thus, the node partitions the data, and the leaf node is the outcome of the prediction.



- The above example shows a decision tree construction using the features that play a significant role in deciding whether to accept a job offer or not.
- The nodes are decided based on **Entropy values** and **Information gain**.
- Entropy is the measure of **uncertainty** of a random variable. It can be calculated using the given formula:

$$\text{Ent}(S) = \sum_{i=1}^{n} -p_i \log_2 p_i$$

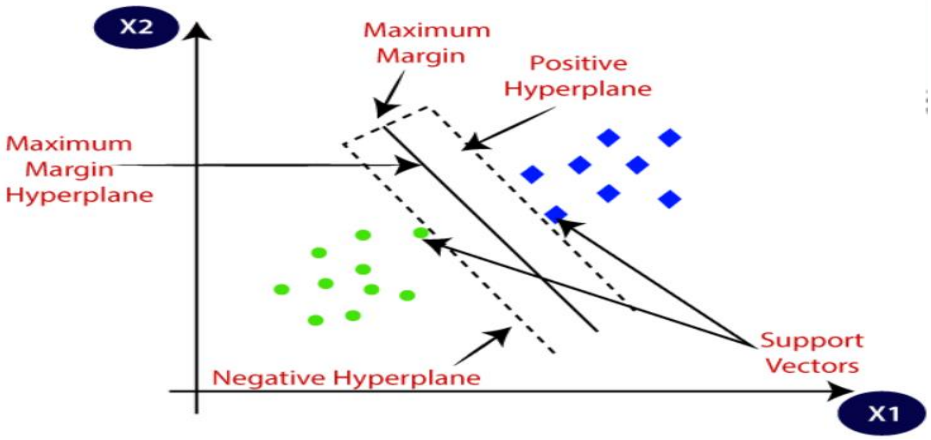
- Where S denotes dataset, n is the number of classes which in the above example is 2 (Yes/No), P(i) is the proportion of class i in the dataset. The min value of entropy can be 0 and the max can be 1.
- Information gain is the **reduction** in entropy from partitioning the data and can be represented as

$$\text{Gain}(S, A) = \text{Ent}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Ent}(S_v)$$

- If the information gain value of the feature is more, then it is considered as the root node, it is also true for the subsequent subtrees.
- The ID3 algorithm works recursively, by first finding the root node and then finding the next root node from the left attributes which becomes the sub-tree. Once the entropy reaches 0 then the tree ends.

SVM (Support Vector Machines):

- The SVM algorithm aims to identify the **hyperplane** which bests classifies the classes. SVM hyperplane can work for n dimensions.
- The hyperplane can be chosen by support vectors. These support vectors are data points that are nearest to the boundary. The hyperplane is chosen such that the distance from these support vectors is the largest. The plane to the right of the boundary is called the positive hyperplane and to the left is called the negative hyperplane.
- For example, if we have to classify Apples and Oranges, then some labeled data will be provided to the SVM model to train itself. The support vectors in this case will be an apple that looks like an orange and an orange that looks like an apple. Based on these vectors, a hyperplane will be generated, then the new data point can be predicted efficiently. The below diagram explains the same.



4. ID3:

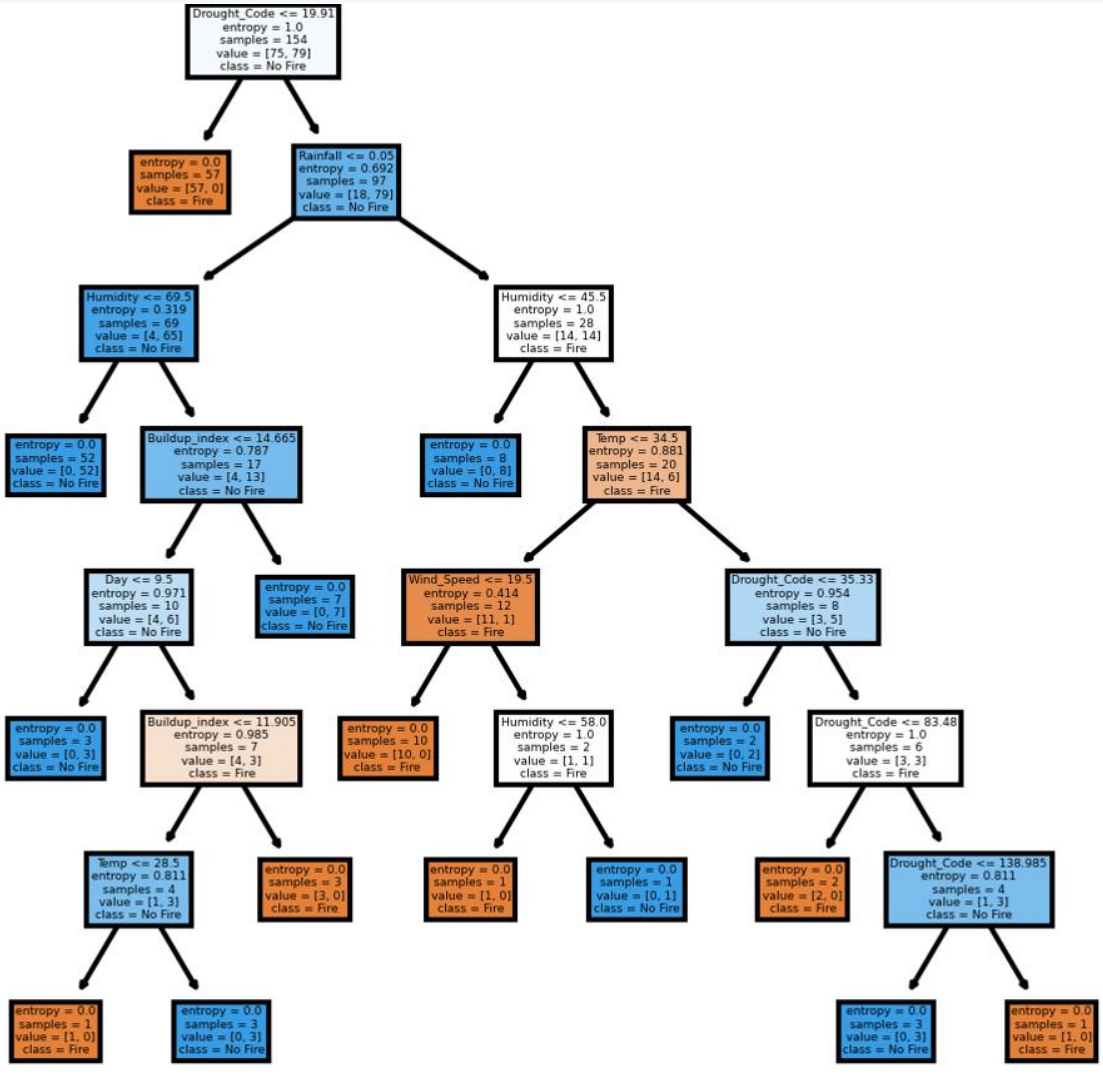
After training the model, I got 86% accuracy for the given dataset.

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import confusion_matrix, accuracy_score
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
print(accuracy_score(y_test, y_pred))
```

```
[[21  1]
 [ 6 22]]
0.86
```

- Below is the decision tree generated for the given data set. It shows that Drought_Code is the root node, which means that it plays an important role in identifying fire. This is calculated using Entropy and Information gain. The higher the information gain, the higher will be the feature in the decision tree. When the entropy reaches 0 then the tree ends, thus denoting whether there will be fire or not which is shown by the class variable in the tree.

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
fn=["Year", "Temp", "Humidity", "Rainfall", "Drought_Code", "Buildup_index", "Day", "Month", "Wind_Speed"]
cn=['Fire', 'No Fire']
tree.plot_tree(classifier,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('decisionTree.png')
```



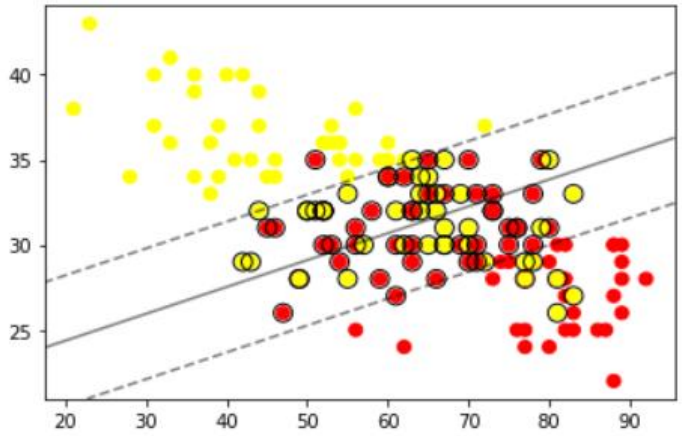
SVM:

After training this model, I got 88% accuracy as shown below:

```
#svm
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score
classifier2 = SVC(kernel = 'linear', random_state = 0)
classifier2.fit(x_train, y_train)
y_pred2 = classifier2.predict(x_test)
cm = confusion_matrix(y_test, y_pred2)
print(cm)
print(accuracy_score(y_test, y_pred2))
```

```
[[20  2]
 [ 4 24]]
0.88
```

- Since the SVM model can be linear and non-linear, I used a linear model to predict whether there will be fire or not.
- In this model, a total of 6 data points was predicted wrong.
- The below scatter plot represents the SVM model having a hyperplane and the support vectors within the plane.



5. The two algorithm shows almost similar results with the accuracy of 86% and 88%. SVM shows higher accuracy because the hyperplane generated is from the nearest support vectors. Hence, any new data that needs to be predicted will have a better chance of classification. For Example, an apple of orange color will be classified correctly in SVM as it will have a support vector with such characteristics.
   In case of the decision tree, the decision is based on nodes. These nodes are identified by Entropy and Information Gain values of the features. Based on these values, sometimes complex data can be classified wrongly. For example, in case of complex data like an apple of orange color maybe classified as an Orange based on its color feature. Hence, the result is different in both models.

References:

1. https://www.nbshare.io/notebook/39408005/SVM-Sklearn-In-Python/
2. https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
3. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
4. https://scikit-learn.org/stable/