Assignment 1

Aim: To understand:

- the basics of understanding the dataset and use respective machine learning category,
- using a machine learning package for a given task
- prepare a short report demonstrating how you have applied it.

Theory/Working:

The data is an important part of this assignment as it represents 4 independent variables (features) and 1 dependent variable (label). The dependent variable contains 2 different classes, setosa and virginica, which is described and differentiated by the features it contains. In future, if we collect random data for the features and we want to know the label, the machine should give us the 'categorical value' of the dependent variable i.e., if it is setosa or virginica. For this, we will use **classification algorithms** from Supervised Learning to categorize the label, which is the type of plant, by looking at features like sepal length, sepal width, petal length and petal width.

There are 2 datasets, plant-train (which contain 80 rows, 40 of setosa and 40 of virginica) and plant-test (which contains 20 rows, 10 for setosa and virginica respectively). To train our model, we can train the model on the plant-train dataset as it contains a good size of data, and we can split the data into 75% training data and 25% data for testing and improving the classification model. We cannot call it an imbalanced dataset as both the plant-train and plant-test dataset contains equal number of rows for each label.

```
df_train['target/label'].value_counts()
✓ 0.1s
setosa       40
virginica    40
Name: target/label, dtype: int64
```

Figure A: Unique count of labels in plant-train.csv

```
df_test['target/label'].value_counts()
✓ 0.1s
setosa       10
virginica    10
Name: target/label, dtype: int64
```

Figure B: Unique count of labels in plant-test.csv

Moving forward, we will use SciKit Learn machine learning package, as it leverages its modules by building on top of several existing Python libraries – NumPy, Pandas, SciPy & Matplotlib, which are equally important as we require Matplotlib and Pandas for collecting data & perform data analysis and visualization. [1] Another advantage is that SciKit learn uses almost similar structure of code formation and output generation across all the algorithms provided. The main features of the package include [2]:

- Easy and free to use.
- Updated by contributors and international online community frequently.
- Provides API documentation for users who want to integrate algorithms to their platforms.

We are going to use 2 classification algorithms i.e., Logistic Regression and Decision Tree Learning on the dataset to find inferences and report them.

We will also use Standard Scaler to scale the data, as it is used for standardization of data The idea behind Standard Scaler is that it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1. In case of multivariate data, this is done feature-wise (in other words independently for each column of the data). Given the distribution of the data, each value in the dataset will have the mean value subtracted, and then divided by the standard deviation of the whole dataset (or feature in the multivariate case) [3].

1. **Logistic Regression:** Logistic Regression is a classification & predictive analysis algorithm which stems from Linear Regression. It's confusing because the name contains the term 'Regression', but it's a proper classification algorithm used:
    a. Categorizing labels using the dataset.
    b. Converting Linear Regression problem to Logistic Regression (making classes using grouping)

In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds, and this logistic function is represented by the following formulas [4]:

$$Logit(pi) = ln(pi/(1-pi)) = Beta_0 + Beta_1*X_1 + \ldots + B_n*X_n$$

Where:

a. **n** is number of features present in the dataset
b. **pi** is the probability the given model.

To train the model, we import plant-train.csv file and perform 75-25% split of train-test data with keeping stratify on the label column so that it helps keep a proportion on the train and test data equally (or nearly equal). We import the logistic regression function from linear model class in SciKit learn and create 2 models to test data with and without normalization of data. Beta coefficients are calculated for each feature and their value lie between -1 to 1. Beta coefficients specify the degree of change in the dependent variable made by each independent variable [5]. Higher the value, higher is the effect of that specific feature on the label. For this algorithm, we have made 2 logistic models, one without the normalization (logistic_model) and one with the StandardScaler() function (logistic_scaled) fitting on the training features, and we get beta coefficients for sepal length, sepal width, petal length and petal width respectively as follow:

```
logistic_model.coef_
✓ 0.9s
array([[ 0.43134826, -0.31597288,  1.65839926,  0.7424394 ]])
```

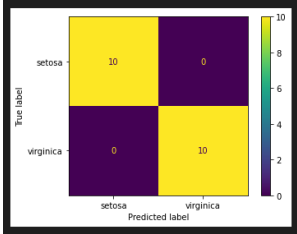
Figure 1.A: Beta Coefficients of features in Logistic Model (Without Normalization)

```
logistic_scaled.coef_
✓ 0.9s
array([[ 0.93447901, -0.72898735,  1.37529256,  1.4081411 ]])
```

Figure 1.B: Beta Coefficients of features in Logistic Model (Standard Scaler Normalization)

For calculating scoring metrics for this algorithm, the accuracy score provides us the model's overall accuracy, and the confusion matrix shows us the categorized values in terms of true positives, false positives, true negatives, and false negatives. These values make us understand the precision, recall and f1-score which makes sense to understand and evaluate our model. To import all of these, we need to use sklearn.metrics class. According to the predictions on the test split we made before on the plant-train.csv, we found these scores on the models in figure 1.C & figure 1.D:

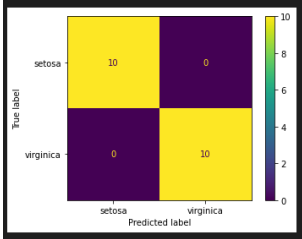
```
print("Accuracy Score: ", accuracy_score(y_test,y_pred1))
✓ 0.4s
Accuracy Score:  1.0

plot_confusion_matrix(logistic_model, X_test, y_test)
✓ 0.1s
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c454fdb3d0>
```




```
print(classification_report(y_test,y_pred1))
✓ 0.4s
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
   virginica       1.00      1.00      1.00        10

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Figure 1.C: Scoring of the test set of plant-train.csv using the "logistic_model" model.

```
print("Accuracy Score: ", accuracy_score(y_test,y_pred2))
✓ 0.3s
Accuracy Score:  1.0

plot_confusion_matrix(logistic_scaled, scaled_X_test, y_test)
✓ 0.1s
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c454fdb370>
```

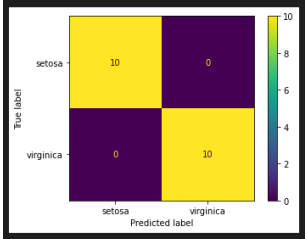



```
print(classification_report(y_test,y_pred2))
✓ 0.3s
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
   virginica       1.00      1.00      1.00        10

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Figure 1.D: Scoring of the test set of plant-train.csv using the "logistic_scaled" model.

After importing the plant-test.csv file, we first test the data on the logistic_model and then we scale the testing data of features with the scaler function and run it via logistic_scaled model. The output when compared to the label in the test dataset is coming as accuracy = 1 for all the metrics as shown in the figure 1.E and 1.F for normal and scaled models respectively.:

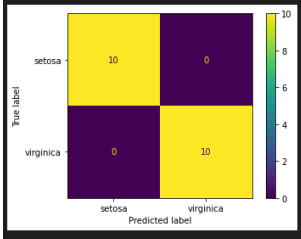
```
print("Accuracy Score: ", accuracy_score(y_test_new,y_pred_new))
✓ 0.3s
Accuracy Score:  1.0

plot_confusion_matrix(logistic_model, X_test_new, y_test_new)
✓ 0.9s
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c454f2e320>
```




```
print(classification_report(y_test_new,y_pred_new))
✓ 0.3s
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
   virginica       1.00      1.00      1.00        10

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Figure 1.E: Scoring of the test set of plant-test.csv using the "logistic_model" model.

```
print("Accuracy Score: ", accuracy_score(y_test_new,y_pred_new2))
✓ 0.3s
Accuracy Score:  1.0

plot_confusion_matrix(logistic_scaled, scaled_X_test_new, y_test_new)
✓ 0.1s
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c455923700>
```




```
print(classification_report(y_test_new,y_pred_new2))
✓ 0.3s
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
   virginica       1.00      1.00      1.00        10

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Figure 1.F: Scoring of the test set of plant-test.csv using the "logistic_scaled" model.

2. **Decision Tree Learning:** Decision Tree Learning is a Supervised Learning technique which can be used for both classification & regression but is mainly preferred for classification solutions. It is based on tree structure, where we have a root node and there are

some decisions to be made from that node to reach another node. The internal nodes represent the feature of a dataset, branches represent the decision rules, and the leaf nodes represents the outcome. [6]

The structure is divided into decision nodes, leaf nodes and a root node. The root node is the first node of the classifier and is also a part of the decision nodes. Decision nodes are the nodes where the decision-making process takes place while the leaf nodes provide the value, or the solution of the decisions taken place. Splitting is a process where one divides the tree into sub-trees based on some given conditions. Pruning is a process of removing unwanted branches from the trees so that the tree doesn't become complex, but not every time you can do pruning or cut down on many branches. Below is a representation of the decision trees:

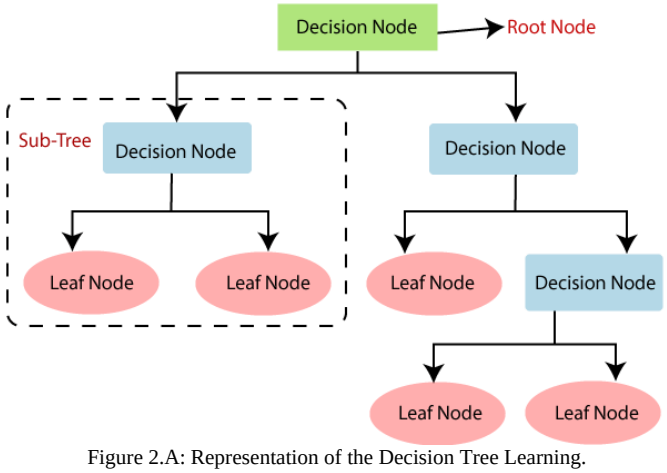


Figure 2.A: Representation of the Decision Tree Learning.

There are 2 attribute selection measures associated with decision trees:

- Information Gain: It is a measurement of changes in entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about the class. Using that value, we split the nodes and build a decision tree. We formulate the information gain as:

  *Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)*

  Where:

  a. Entropy is *Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)*
  b. S= Total number of samples
  c. P(yes)= probability of yes
  d. P(no)= probability of no

- Gini Impurity: Gini Impurity is the measure of purity or impurity used when creating a decision tree algorithm. An attribute with the low Gini index should be preferred compared to a high Gini index. It can only create binary splits, and so its usually preferred with Classification and Regression Tree (CART) algorithm. The formula is as follows:

  Gini Index= 1- ∑jPj2

  Where: **P** is the probability of the given number.

To train the model, let's import the plant-train.csv file and split it into 75-25% training-testing datasets. We import DecisionTreeClassifier function from the sklearn.tree class. We make 2 models i.e., dt_model for creating a model without any data normalization, and dt_scaled for creating a model with data normalization. Generally, the decision trees don't require normalization but as we are suggested, we will use the basic StandardScaler() to scale the data points. For calculating metrics for this algorithm, confusion matrix works best to see how the model is predicting the class. Also, use of classification report provides us with how the model scores are being calculated.

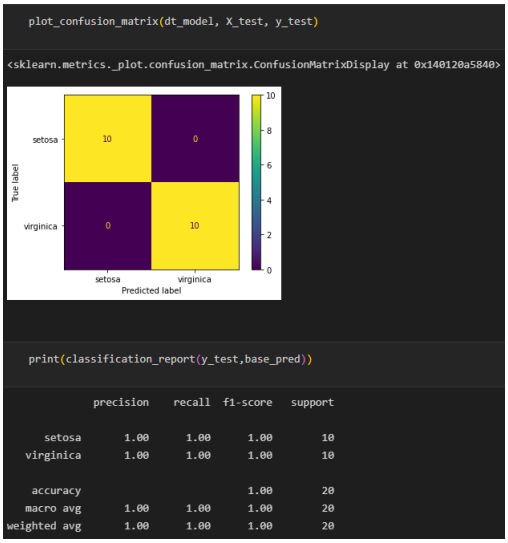


Figure 2.B: Scoring of the test set of plant-train.csv using "dt_model" model.

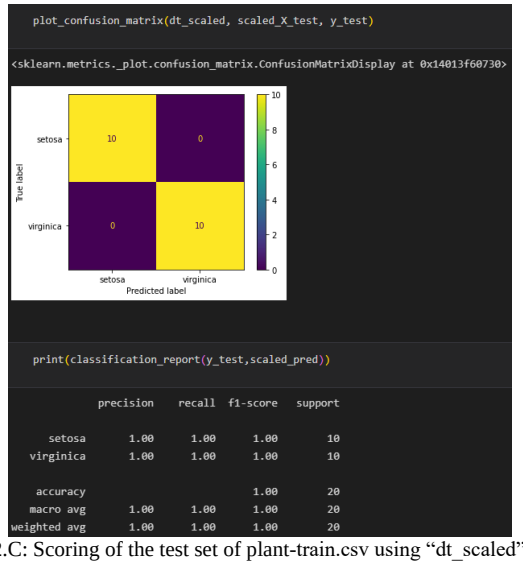


Figure 2.C: Scoring of the test set of plant-train.csv using "dt_scaled" model.

```
pd.DataFrame(index = X.columns, data = dt_model.feature_importances_, columns = ['Feature Importance'])
```

	Feature Importance
sepal length in cm	0.0
sepal width in cm	0.0
petal length in cm	0.0
petal width in cm	1.0

```
from sklearn.tree import plot_tree

plot_tree(dt_model, feature_names = X.columns, filled = True)
```

```
[Text(0.5, 0.75, 'petal width in cm <= 0.9\ngini = 0.5\nsamples = 60\nvalue = [30, 30]'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 30\nvalue = [30, 0]'),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 30\nvalue = [0, 30]')]
```

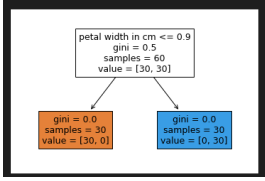


Figure 2.D: Feature importance and decision tree layout derived from the "dt_model" model.

```
pd.DataFrame(index = X.columns, data = dt_scaled.feature_importances_, columns = ['Feature Importance'])
```

	Feature Importance
sepal length in cm	0.0
sepal width in cm	0.0
petal length in cm	1.0
petal width in cm	0.0

```
plot_tree(dt_scaled, feature_names = X.columns, filled = True)
```

```
[Text(0.5, 0.75, 'petal length in cm <= -0.157\ngini = 0.5\nsamples = 60\nvalue = [30, 30]'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 30\nvalue = [30, 0]'),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 30\nvalue = [0, 30]')]
```

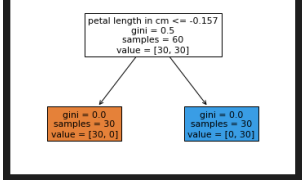


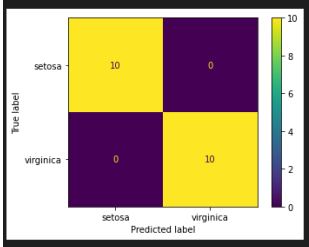
Figure 2.E: Feature importance and decision tree layout derived from the "dt_scaled" model.

An interesting finding from looking at Figure 2.D and 2.E is that when we calculated our decision trees on the dt_model, it gave 1.0 importance to the feature 'petal width in cm', but it gave a different answer for dt_scaled, where it calculated 1.0 importance to 'petal length in cm'. The decision tree calculated Gini index =0.5 as the root node value in dt_model for petal width in cm <=9 and when the decision was made, the Gini index went to 0.0 in both the leaf nodes. Whereas for the dt_scaled, it calculated Gini index = 0.5 for the petal length in cm <=0.157 (the value decreased as we scaled the features for this model) for the root node, and it calculated Gini index as 0.0 in both the leaf nodes. Thus, we observe that scaling the model to be effective here as after all the features got scaled to values between 0 and 1, the actual factor which was considered in dt_scaled was petal length and not the width as we saw in the dt_model. Both the models performed similar as we see the classification report and confusion matrix returning us the accuracy of 1.

Finally, let's check our data from the plant-test.csv where we import and split the data. We test the features without data normalization and then using the data normalization and as we saw above the feature importance in Figure 2.D and 2.E, we found inferences of the metrics as:

```
plot_confusion_matrix(dt_model, X_test_new, y_test_new)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14013ac9e70>
```



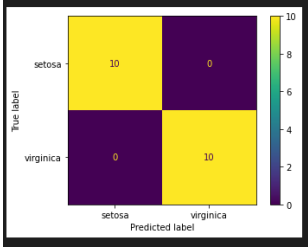

```
print(classification_report(y_test_new,y_pred_new))
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
virginica	1.00	1.00	1.00	10
accuracy			1.00	20
macro avg	1.00	1.00	1.00	20
weighted avg	1.00	1.00	1.00	20

Figure 2.F: Scoring of the test set of plant-test.csv using the "dt_model" model.

```
plot_confusion_matrix(dt_scaled, scaled_X_test_new, y_test_new)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x140141834f0>
```




```
print(classification_report(y_test_new,y_pred_new2))
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
virginica	1.00	1.00	1.00	10
accuracy			1.00	20
macro avg	1.00	1.00	1.00	20
weighted avg	1.00	1.00	1.00	20

Figure 2.G: Scoring of the test set of plant-test.csv using the "dt_scaled" model.

Conclusion: We infer that the models presented in this assignment worked well to prove us that the dataset contained good categorization feature set to get an accuracy of 1 while predicting the labels for both Logistic Regression and Decision Tree Learning classification algorithms. Although we get the best prediction accuracy in both the classifications, as Logistic Regression has used majority the features as per the beta coefficients to calculate and predict the class label, while Decision Tree Learning used Gini Impurity and only used either petal width in normal data, and petal length in scaled data to calculate and predict the class label. This distinction lets us understand that the algorithms have determined their importance of features in the data set to predict accurately the class labels and have their distinct methods and calculations to find out the predictions of an unknown dataset.

References

[1] S. Yegulalp, “14 open source tools to make the most of machine learning,” Info World, 23 September 2020. [Online]. Available: https://www.infoworld.com/article/3575420/14-open-source-tools-to-make-the-most-of-machine-learning.html.

[2] B. Kumar, “What is Scikit Learn in Python,” Python Guides, 10 December 2021. [Online]. Available: https://pythonguides.com/what-is-scikit-learn-in-python/.

[3] “Can anyone explain me Standard Scaler?,” Stack Overlow, [Online]. Available: https://stackoverflow.com/questions/40758562/can-anyone-explain-me-standardscaler.

[4] “What is Logistic Regression?,” IBM, [Online]. Available: https://www.ibm.com/topics/logistic-regression.

[5] G. Choueiry, “Interpret Logistic Regression Coefficients [For Beginners],” Quantifying Health, [Online]. Available: https://quantifyinghealth.com/interpret-logistic-regression-coefficients/.

[6] “Decision Tree Classification Algorithm,” Java T Point, [Online]. Available: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm.