

1. **Answer:** Given data contains four independent variables, namely sepal length, sepal width, petal length and petal width. These variables determine value of our dependent variable "target/label". Target/label contains two values setosa and virginica. These are two types of flowers, and the dependent variables are features that determine the type of the flower. In the given dataset, the target label is given; hence, it is a case of supervised learning problem. Moreover, the target label is a discrete variable as it contains two values setosa and virginica, hence it is a classification task.
2. **Answer:** Datasets that contain uneven distribution of target class variables are called imbalanced dataset.

```
In [4]: print(train_data["target/label"].value_counts())
        print(test_data["target/label"].value_counts())

setosa      40
virginica    40
Name: target/label, dtype: int64
setosa      10
virginica    10
Name: target/label, dtype: int64
```

In both the training and testing datasets, the target class variables have an even distribution of setosa and virginica observations. Thus, the dataset is not imbalanced.

3. **Answer:** There are a plethora of open-source machine learning packages available for python free of cost. For this task we have used packages like NumPy [1] a python package that is useful to transform your data in matrix form that later could be used for performing mathematics operations. Pandas [2] a package that is used for data manipulation purposes. For machine learning tasks there are various libraries like Scikit-learn, TensorFlow, PyTorch and Keras. Scikit-learn is the chosen package for this task. Implementing machine learning models for small datasets is simple in Scikit-learn. TensorFlow, PyTorch and Keras are generally used to develop highly complex deep learning models.

Brief Overview of Sci-Kit-learn package [3]:

Sci-Kit contains various sub packages that are meant to perform specific tasks

1. `sklearn.preprocessing` : this sub package contains various functions for cleaning and standardizing the dataset for machine learning task these include standardizing functions like `StandardScaler`, `MinMaxScaler`.
2. `sklearn.model_selection`: It contains functions used to create train, test and validation sets like `kfold`, `train_train_split`.
3. `sklearn.metrics` : Metrics that are used to validate the machine learning models like accuracy score, confusion matrix, `f1_score` are contained in this sub package.
4. Machine learning models: There are various sub packages that contain machine learning models like `sklearn.linear_model` that contains models like Logistic Regression, Linear Regression also for models defined on Naïve Bayes there is a package `sklearn.naive_bayes` it contains models like Multinomial, GaussianNB and CategoricalNB. `Sklearn.neighbors` contains models for Nearest Neighbours algorithms like Nearest Neighbours, `KNearestClassifier`, `KNearestRegressor`.
4. **Answer:** Normalisation for the data is done using `MinMaxScaler` function from `sklearn.preprocessing` package. In Normalisation we transform our data between a certain range of values in this case it is between 0 and 1.

For the second model data standardisation is used for preparing data. In Standardisation we transform the data such that mean of the data samples is 0 and standard deviation is zero. In sci-kit learn standardisation can be done by using `StandardScaler` function from `sklearn.preprocessing` library.

5. **Answer:** Two machine learning algorithms K-nearest-neighbour and Logistic regression are used for this dataset. The task is of classification type and K-nearest-neighbour and logistic regression both are classification algorithms. Moreover, KNN is a lazy learner and logistic regression is an eager learner. During training phase, a lazy learner just stores the data without learning it and classification of data points are done during testing phase. On a

contrary, eager learners learn during the training phase. Consequently, eager learners take more time on training compared to testing and lazy learners take more time in testing phase.

K – Nearest Neighbour on the dataset[4]:

1. Apply standardisation on both testing and training data. StandardScaler transforms data such that mean of data = 0 and standard deviation = 1.

```
In [6]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
x_test = scaler.transform(x_test)
```

2. Apply KNearestNeighbours on standardised dataset. In KNN first we need to set value of hyperparameters. We take the value of K = 3 and metric as Euclidean distance.
3. KNN thinks of every training case in training set as a datapoint in space. And during the testing phase it projects testing case in this hyperspace.
4. The distance of all the datapoints in the training set are calculated against testing case. As the value of K is 3, three nearest datapoints are considered.
5. The output of the target variable depends on the target variable value of these three nearest datapoint. Average of three labels is considered and assigned as output.
6. For example, if the target variable of three datapoints with the least Euclidean distance to the testing datapoint are ['setosa', 'virginica', 'setosa'] then setosa would be assigned as target variable for the testing case.

Logistic Regression [5]:

1. Apply normalisation on both testing and training data. StandardScaler transforms data such that range of variables are between range of 0 and 1.

```
In [6]: from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
x_test = scaler.transform(x_test)
```

2. Logistic regression is a classification algorithm. In this example we are doing to use logistic regression to predict target label which belong to either of the following values ['setosa', 'virginica'].
3. Just like in linear regression, we need to train a hypothesis in logistic regression. In this case hypothesis of logistic regression would be.

$$\bar{y} = m_1x_1 + m_2x_2 + m_3x_3 + m_4x_4 + c$$

Here y is the target/label

x_1, x_2, x_3, x_4 are independent variables

And m_1, m_2, m_3, m_4 are slopes for the independent variables that need to be calculated

c is the intercept.

4. Now in regression we try to minimize the error rate of the hypothesis i.e the value predicted by hypothesis \bar{y} and actual value y should be minimized denoted by cost function.

$$\text{minimize cost} = \frac{1}{2n} \sum_{i=1}^n \bar{y}_i - y_i$$

where n is number of terms in training set

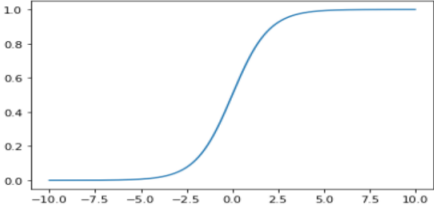
5. Minimizing the cost is done by using gradient descent algorithm. Which minimizes the cost while iterating over the training data. The output Y is right now a continuous value it can be converted to binary form by using sigmoid function. Sigmoid/Logistic function basically converts any input value to an output that is between range of 0 and 1.

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

```
In [18]: import numpy as np
def sigmoid(x):
    return 1/(1+np.exp(-x))

In [19]: x = np.linspace(-10, 10, 1000)
y = sigmoid(x)

import matplotlib.pyplot as plt
plt.plot(x, y)
plt.show()
```



After training is completed, the hypothesis is tested on unseen testing data to check the accuracy of the model trained. Accuracy achieved for KNN, and Logistic regression is same with or without normalisation of data. Reasoning is explored in question 6 and 7

6. **Answer:** Training data and testing data were normalised (for Logistic Regression) and standardised(for K-Nearest Neighbours). Now, the data needs to be trained on machine learning models for KNN and logistic regression from scikit-learn library.

Training on KNN and Logistic Regression

Hyper- Parameters for K-Nearest Neighbour: Hyperparameter: K = 3; distance measure = “Euclidean”.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3, metric='euclidean')

knn.fit(X_train, Y_train)

KNeighborsClassifier(metric='euclidean', n_neighbors=3)

y_pred = knn.predict(x_test)
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()

lr.fit(X_train, Y_train)

LogisticRegression()

y_pred = lr.predict(x_test)
```

Performance Metrics for algorithms

For: K-Nearest Neighbour

```
In [13]: from sklearn.metrics import accuracy_score, confusion_matrix

In [11]: accuracy_score(y_test, y_pred)

Out[11]: 1.0

In [14]: confusion_matrix(y_test, y_pred)

Out[14]: array([[10,  0],
               [ 0, 10]], dtype=int64)
```

For: Logistic Regression

```
In [10]: from sklearn.metrics import accuracy_score, confusion_matrix

confusion_matrix(y_test, y_pred)

Out[10]: array([[10,  0],
               [ 0, 10]], dtype=int64)

In [11]: accuracy_score(y_test, y_pred)

Out[11]: 1.0
```

For both the algorithms we get all the test samples classified correctly, In the confusion matrix we observe that 10 cases were classified as true positive and 10 as false negative for both the algorithms.

For K-Nearest Neighbour

```
In [15]: print(classification_report(y_test, y_pred))
```

	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

For Logistic Regression

```
In [25]: from sklearn.metrics import classification_report

print(classification_report(y_test,y_pred))
```

	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

For both KNN and Logistic Regression Precision and Recall scores for both the classes virginica and setosa are a perfect 1.0. Precision score of 1 means that every item labelled as belonging to a certain target variable does belong to that target. But doesn't say anything about target variables that were not labelled correctly.

Whereas a Recall score of 1 means that every item labelled as every item belonging to a certain target class was classified correctly but says nothing about how many items from other class were also incorrectly labelled to be of that target class.

7. Answer:

```
In [22]: # Logistic regression
%%timeit
lr.predict(x_test)
```

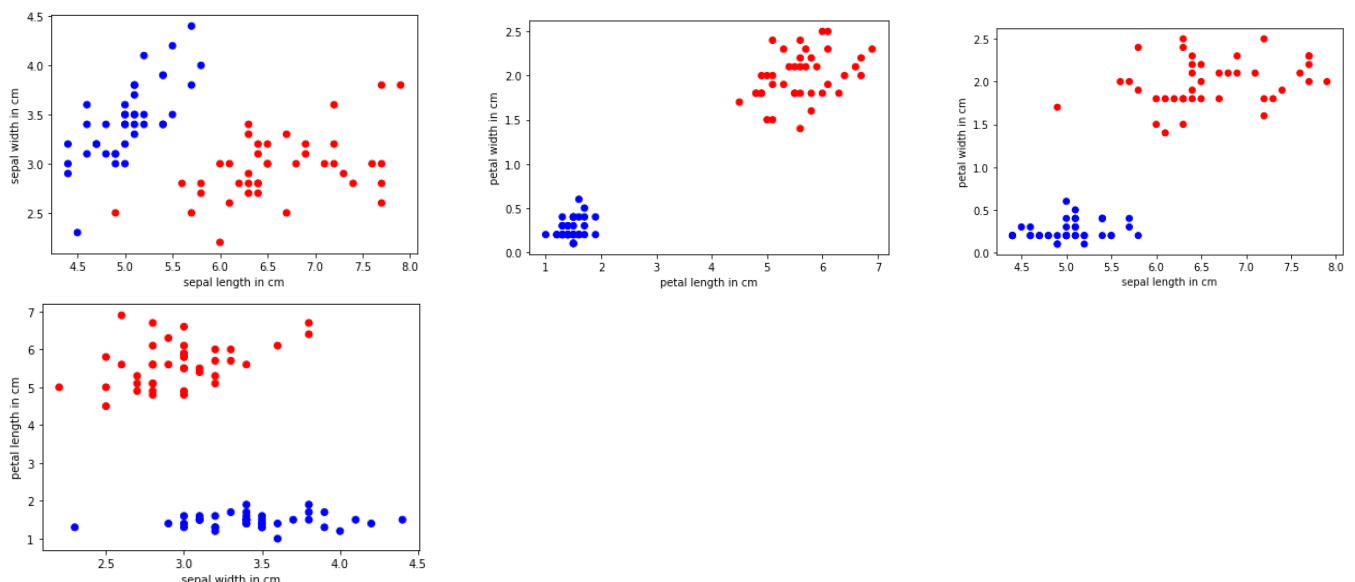
29 μ s \pm 1.12 μ s per loop (mean \pm std. dev. of 7 runs, 10000 loops each)

```
In [20]: #K-NearestNeighbour
%%timeit
knn.predict(x_test)
```

556 μ s \pm 12.1 μ s per loop (mean \pm std. dev. of 7 runs, 1000 loops each)

As you can see by runtime logistic regression is much faster in testing phase than K-Nearest Neighbour as KNN is a lazy learner it learns during the testing phase whereas in logistic regression hypothesis is formed during the training phase.

Although the time taken by KNN and Logistic Regression for generating output differs, they are performing well on testing data. Testing data is unseen, yet they both predict samples in testing dataset with high accuracy. This maybe because the independent variables of the data are highly segregated and thus classifier trains better on identifying the target label. Also, the size of dataset is really small for the algorithm to work, more samples are needed to train and test so that the trained ML model is robust. In the below scatter plots red are virginica samples and blue are setosa samples.



References:

- [1] <https://numpy.org/doc/>
- [2] <https://pandas.pydata.org/docs/>
- [3] <https://scikit-learn.org/stable/>
- [4] CT5165 Principles of Machine Learning : Week 4 slides (Similarity based learning)
- [5] <https://towardsdatascience.com/quick-and-easy-explanation-of-logistics-regression-709df5cc3f1e>