1920-CT4101 Machine Learning
Name: SWATI Student ID: 19233301 Class:1MAI1

Algorithm selected for the assignment is

Logistic Regression:

This is implemented in scikit-learn from scratch, programming language used is python and the algorithm is also tested as a part of the assignment.

Logistic regression is a classification method (Supervised Learning) of machine learning and is usually used to classify binary classes. Logistic Regression predicts something is TRUE/FALSE, instead of predicting something continuous like size.

The output given will be either 0 or 1, which says whether the target variable belong to class 0 or class 1

The function that outputs the probability of the class is the hypothesis (h) function, which returns the probabilities with numbers between 0 and 1. Before h, we need to formulate the Logistic function which is also referred to as sigmoid function.

- The Logistic function is,
  Logistic(t) = $1 / 1 + e^{-t}$
  t = theta * x
- The hypothesis function is,
  h(x) = $1 / 1 + e^{-\text{theta} * x}$
  Where, x is the weights
  If "t" goes to +infinity, Y (predicted) will become 1 and if "t" goes to –infinity, Y (predicted) will become 0. The hypothesis gives a soft boundary with a probability of 0.5 at the centre of the boundary region, and approaches 0 or 1 as we move away from the boundary.
- The Loss function is,
  J(theta) = $1 / m * (-y^{\text{theta}} * \log(h) - (1-y)^{\text{theta}} \log(1-h))$
  This function gives the best value for weights, here theta. It shows the value of how well the algorithm has performed.
- The Gradient descent function,

$$\frac{\partial}{\partial \theta_j} J(\theta) = \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$$

#Formula taken from https://towardsdatascience.com/gradient-descent-demystified-bc30b26e432a

It is an iterative method which finds the minimum of loss function. With every iteration, the weights (theta) gets updated. The best values of theta are used for prediction of the result.

There are two types of classification in logistic regression,

1. *Binary Classification*:
   When the target variable has 2 classes that needs to be predicted. For example, mails classified into spam or not spam. There are only 2 classes either positive class or negative class and the values will either be 1(belong to the class) or 0(does not belong to the class).
2. *Multi Class Classification:*
   When the target variable has more than 2 classes that should be predicted. For example, the hazelnuts data the target variable has 3 classes, 'c_americana', 'c_avellana' and 'c_cornuta'. Here, each class is treated as a binary classification and we

predict 1 class with other 2 classes. The approach for multi class classification followed here is "One vs Rest" where we train the classifier with 3 classes and predict once by taking the maximum of the probabilities returned by classifier.

SOURCE:

Artificial Intelligence A Modern Approach Third Edition Stuart J. Russell and Peter Norvig

https://www.coursera.org/lecture/machine-learning/multiclass-classification-one-vs-all-68Pol

https://en.wikipedia.org/wiki/Logistic_regression

https://medium.com/@martinpella/logistic-regression-from-scratch-in-python-124c5636b8ac

Design Decisions:

I have used one vs Rest approach for multiclass classification, Firstly, I will be importing hazelnuts.csv. Before importing data, I have done some data pre-processing in an excel sheet (Mainly transpose of rows to columns). Once imported, I am putting all the target (dependant) variables into X and all the feature (independent) variables into y.

To split the data into 2/3 for training and 1/3 for testing I am using the "train_test_split" method, which will divide both X and y into X_train, X_test, y_train & y_test. Using StandardScaler for normalising the data. StandardScaler will transform X_train & X_test such that its distribution will have a mean value 0 and standard deviation of 1. This will help in training the model better resulting in more accurate predictions.

#Def of StandardScaler taken from https://stackoverflow.com/questions/40758562/can-anyone-explain-me-standardscaler

To apply one vs Rest method, I have divided train data, i.e. splitting y_train into (0's/1's), y1_train, y2_train & y3_train where y1_train is marked 1 only for 'c_americana' and rest are 0, y2_train is marked 1 only for 'c_avellana' and rest are marked 0, y3_train is marked 1 only for 'c_cornuta' and rest are 0. This way the model is trained for all 3 classes.

Class Logistic Regression has 5 methods,

- ***def init*** refers to the newly created object in the class, and I am initializing 2 parameters, alpha which is the learning rate and num_iter is the number of iterations.
- ***def sigmoid*** returns the hypothesis, it takes only 1 value which is (z) the dot product of X and w.
- ***def fit*** takes the attributes which are the 10 columns, and w which are the weights. Z is the dot product of $x^1 * w^1$. The gradient descent for each value of w is calculated by the formula,
  w = w - alpha * sum ((h - y) * X [: , j]). The objective of gradient descent is to find out optimal parameters that result in optimising the function. The value of w gets updated for each iteration, and 10000 such iterations are run to get the best possible values for weights.
- ***def predict_proba*** returns the probabilities for all the 3 trained classes.
- ***def predict*** returns the predictions. It takes the maximum of probabilities returned by predict_proba

SOURCE:

https://www.coursera.org/lecture/machine-learning/multiclass-classification-one-vs-all-68Pol

Code flow:

To run the logistic regression algorithm,
First we call LogisticRegression model, and fit method. In fit we train the model for 3 classes, y1_train, y2_train and y3_train. Then, we call the predict method, which will return predictions in strings and it is easy for us to compare it against the true values.
The accuracy score is calculated by true values and the predicted values.
Finally, the true values, predicted values along with the output (true/false) is written to a csv file.

Testing: (Comparing with reference implementation)

MY MODEL

In order to test the model, I am splitting the data with 10 random divisions with shuffle set to TRUE. This happens inside a for loop which will run 10 times and calculate the accuracy for each run/fold. By doing this our model will be trained and tested for random splits of data and the performance can be mapped better. X_train, X_test, y_train & y_test is different for each split and data is shuffled.

Below are the results given by my model on 10 random splits and the accuracy scores of each split along with the average accuracy score is computed.

```
Accuracy score of split  1 :  94.02985074626866
Accuracy score of split  2 :  94.02985074626866
Accuracy score of split  3 :  91.04477611940298
Accuracy score of split  4 :  92.53731343283582
Accuracy score of split  5 :  89.55223880597015
Accuracy score of split  6 :  91.04477611940298
Accuracy score of split  7 :  95.52238805970148
Accuracy score of split  8 :  92.53731343283582
Accuracy score of split  9 :  92.53731343283582
Accuracy score of split  10 :  85.07462686567165

Average Accuracy score of model: 91.7910447761194
```

SCIKIT LEARN MODEL

For comparing the implemented algorithm, I have interpreted the results of my model with Logistic Regression algorithm already implemented on scikit learn. To achieve this, I have imported LogisticRegression in scikit learn. Since, I used ***StandardScaler*** to normalize data, I used it here as well. Calculating the cross_val_score of 10 folds. Here, at every run the data is split internally as train and test and this continues for 10 folds.

Below are the results given by scikit learn model on 10 fold cross validation and the accuracy scores of each fold along with the average accuracy score is computed.

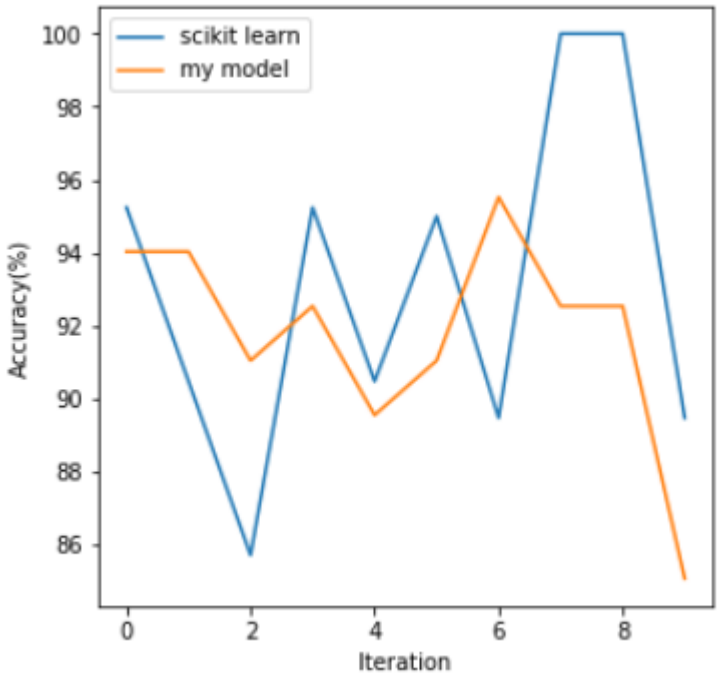
```
Accuracy score of cross validation  1 :  95.23809523809523
Accuracy score of cross validation  2 :  90.47619047619048
Accuracy score of cross validation  3 :  85.71428571428571
Accuracy score of cross validation  4 :  95.23809523809523
Accuracy score of cross validation  5 :  90.47619047619048
Accuracy score of cross validation  6 :  95.0
Accuracy score of cross validation  7 :  89.47368421052632
Accuracy score of cross validation  8 :  100.0
Accuracy score of cross validation  9 :  100.0
Accuracy score of cross validation  10 :  89.47368421052632

Accuracy score of scikit learn model: 93.10902255639097
```

LEARNING CURVE

A learning curve shows the performance of a machine learning algorithm, it is plotted as rate of its learning against experience. It can be very handing when we want to compare the performance of 2 different algorithms, and to identify if the model has well-fit, underfit or overfit model.

Here, I have plotted a Performance Learning Curve. It compares the performance (accuracy) of the 2 models for 10 iterations.



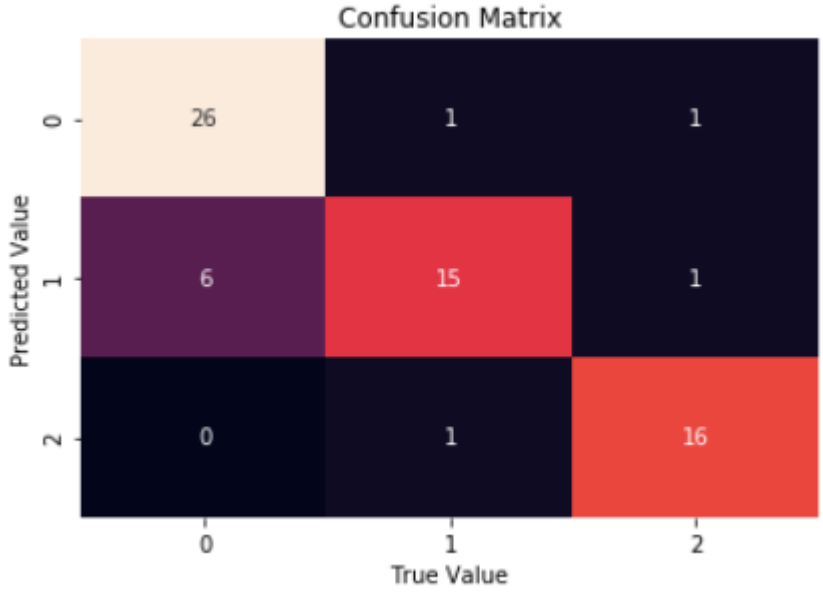
Analysis:
The implemented model seem to be doing slightly better than the one implemented by me on scikit learn. Performance of the models are hand in hand good. Accuracy scores of my model start from 94% then, the learning slightly comes down to about 91%. However, again at the 7th iteration the performance increases to 95%.

Whereas, the scikit learn (reference) model starts off by 95% and then the learning falls to 85% on the 3rd iteration. The model picks up the score by 100% accuracy on the 8th & 9th iteration.

SOURCE:
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance
https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html

Conclusions & Observations:

For observations, I have plotted a confusion matrix for my model which will help in giving a summary of the entire classification model. It is computed using the true and predicted values of classification. It will tell us how many right predictions the model did make and the errors that occurred during classification.
Here, we have 3 classes so the confusion matrix will be a 3X3 matrix with the values for all 3 classes “c_americana”, “c_avellana” & “c_cornuta”



As seen above, class 0 -> c_americana
class 1 -> c_avellana
class 2 -> c_cornuta
Total of 67 samples are shown in the confusion matrix, it is the y_test and y_pred which are 1/3 of 201 instances of the whole dataset
For class 0 my model has predicted 26 true positives, 6 false negatives
For class 1 my model has predicted 15 true positives, 1 false negative & 1 false positive
For class 2 my model has predicted 16 true positives, 1 false negative & 1 false positive

“Both algorithms perform similar, because they are same algorithms coded on python only the implementation is different. They make use of one vs rest for their classification, we can perform binary as well as multiclass classification on logistic regression. Accuracy of both the models are more or less the same. They have performed well and it can be seen from the performance learning curve. The reference model runs faster than mine, because of the number of iterations that we give and it also depends on the learning rate”

SOURCE:https://machinelearningmastery.com/confusion-matrix-machine-learning/

Appendix:

```
1.  import csv
2.  import numpy as np
3.  import pandas as pd
4.  import matplotlib.pyplot as plt
5.  from sklearn.model_selection import train_test_split
6.  from sklearn.metrics import accuracy_score
7.  from sklearn import metrics
8.  from sklearn.preprocessing import StandardScaler
9.  from sklearn.model_selection import cross_val_score
10. from sklearn.metrics import confusion_matrix
11. import seaborn as sns
12. import warnings
13. warnings.simplefilter("ignore")
14.
15. #importing csv file
16. hazelnut = pd.read_csv(filepath_or_buffer  = "C:\MSc-
    Artificial Intelligence\Semester 1\Machine Learning\Assignments\hazelnut1.csv", sep
     = ',', header = None)
17.
18. hazelnut.head()
19.
20. #seperating target & feature variables
21. X = hazelnut.iloc[:,:-1].values
22. y0 = hazelnut.iloc[:,-1].values
23.
24. class LogisticRegression(object):
25.     def __init__(self, alpha=0.01, num_iter=10000):
26.             self.alpha = alpha
27.             self.num_iter = num_iter
28.
29.     def sigmoid(self, z):
30.         sig = 1 / (1 + np.exp(-z))
31.         return sig       #returns probabilities
32.
33.
34.     def fit(self, X, y):
35.         self.num_attributes = X.shape[1]
36.         self.w = np.zeros((X.shape[1],))
37.
38.         for i in range(self.num_iter):
39.             z= np.dot(X,self.w)
40.             h = self.sigmoid(z)
41.             for j in range(self.num_attributes):           #number of columns in X

42.                 self.w[j] = self.w[j] - self.alpha * np.sum((h - y) * X[:,j])     #
    gradient descent
43.
44.         return self.w    #returns the best value for w
45.
46.     def predict_prob(self, X, w):
47.         z1 = np.dot(X, w.T)
48.         return self.sigmoid(z1)   #returns value from sigmoid function
49.
50.     def predict(self, X, w):
51.         h_pred = self.predict_prob(X, w)
52.         y_max = np.argmax(h_pred, axis = 1)   #gives the index value of maximum pro
    bability
53.         Predictions = []
54.         for i in y_max:                       #converts 0's, 1's to strings of clas
    s name
55.             if i == 0:
```

```
                Predictions.append("c_americana")
            elif i == 1:
                Predictions.append("c_avellana")
            elif i == 2:
                Predictions.append("c_cornuta")
        return Predictions                    #returns predicted values

Accuracy_10_fold = []
for i in range(10):  #splits data into 10 random divisions
    X_train, X_test, y_train, y_test = train_test_split(X, y0, test_size = 1/3, shuffle=True)
    #splits train data into 67% and test data into 33%

    ss = StandardScaler() #normalize the data

    X_train = ss.fit_transform(X_train)
    X_test = ss.transform(X_test)

    #one vs rest
    y1_train = np.where(y_train == 'c_americana', 1, 0)
    y2_train = np.where(y_train == 'c_avellana', 1, 0)
    y3_train = np.where(y_train == 'c_cornuta', 1, 0)

    weights = np.zeros((3,10))
    clf = LogisticRegression()

    weights[0] = clf.fit(X_train, y1_train)  #training for 3 classes
    weights[1] = clf.fit(X_train, y2_train)
    weights[2] = clf.fit(X_train, y3_train)

    y_pred = clf.predict(X_test, weights)   #predicting
    Accuracy = accuracy_score(y_test, y_pred)

    Accuracy_10_fold.append(Accuracy*100)  #compute accuracy score for each split

    print("Accuracy score of split ", i+1, ": ", Accuracy_10_fold[i])

Avg_Accuracy = np.mean(Accuracy_10_fold)  # returns mean of all the scores
print("\nAverage Accuracy score of model:", Avg_Accuracy)

# Confusion Matrix
matrix = confusion_matrix(y_test, y_pred)

#heatmap
plot1 = sns.heatmap(matrix,annot=True,cbar=False)
        plt.ylabel('Predicted Value')
        plt.xlabel('True Value')
        plt.title('Confusion Matrix')
        plt.show()

        # Reference logistic Regression to compare model

        from sklearn.linear_model import LogisticRegression

        ref = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomial').fit(X, y0)
        ref.predict(X)
        X = ss.fit_transform(X)
        for i in range(10):
            scores = cross_val_score(ref, X, y0, cv=10) * 100
            print("Accuracy score of cross validation ", i+1, ": ", scores[i])
        score1 = np.mean(scores)
        print("\nAccuracy score of scikit learn model:", score1)

        #learning curve
        plt.figure(figsize=(5,5))
```

```
        plt.plot(range(10), scores, label = 'scikit learn')
        plt.plot(range(10), Accuracy_10_fold, label = 'my model')
        plt.ylabel('Accuracy(%)')
        plt.xlabel('Iteration')
        plt.legend()
        plt.show()

        #writing to csv

        predictions = list()
        for i in range(len(y_pred)):
            predictions.append([y_test[i], y_pred[i], "True" if y_pred[i] == y_test[
    i] else "False"])
            Predict = pd.DataFrame(predictions, columns=["True Value", "Predicted Va
    lue", "Output"])

        Predict.to_csv('LogisticRegression.csv', header=True, index=False)
```