Assignment 3
Logistic Regression – ML

Niranjanadevi J-18230368
Shruthi Sundhar-18230295

IMPLEMENTATION AND ALGORITHM DETAILS

Logistic Regression is one of the popular algorithms to solve a classification problem. Logistic regression it is an extension of linear regression, mostly the dependent variable is categorical and not continuous. The probability of the outcome variable is predicted by this algorithm.

MULTI-NOMINAL REGRESSION

In binary or binomial logistic, the outcome has two values (e.g. either 'Yes' or 'No', 'Success' or 'Failure'). In binary logistic regression, the outcome is coded as '0' and '1', but Multinomial logistic generally have three or more outcomes (e.g.: 'Good', 'Very good', 'Best'). Since our dataset 'owl.csv' have 3 types of predictors (Owl Types) we are performing multinomial regression algorithm to our dataset.

LOGISTIC FUNCTION(SIGMOID FUNCTION)

In the logistic regression algorithm, a linear equation with independent predictors is used to predict the values, which range between negative infinity to positive infinity. The output of algorithm a class variable, i.e. 0 or 1. To determine the predicted value between 0 to 1, a sigmoid function is used.

$$z = \theta_0 + \theta_1 \cdot x_1 + \theta \cdot x_2 + \cdots$$

Linear Equation

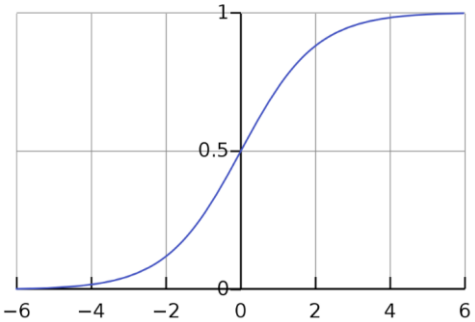
$$g(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid Function

When we pass the output(z) value to g(x) function it gives value 'h' where the values range from 0 to 1.

$$h = g(z) = \frac{1}{1 + e^{-z}}$$

When plotting the graph, we get a S curve where the sigmoid function becomes close to arbitrary y.



COST FUNCTION:
The logarithmic loss function is used to calculate the cost for misclassified values

$$\frac{\partial J}{\partial \theta_n} = \frac{1}{m} \cdot x_i \cdot [\sum_{i=1}^{m} h_i - y_i]$$

ADVANTAGES:
It is an efficient technique and widely used one. Too many computations is not required and also don't require scaled input features and easy regularization. Predicted probabilities outputs are well calibrated.

ALGORITHM DESIGN:

Language Used: Python
Platform: Google Colaboratory

Data-Loading, Assigning and attributes:

The file owls.csv is loaded and the dataset is separated into features and predictors. In the given data set the predictor is of three types BarnOwl, and SnowyOwl. There are four features in the given data set Body-length, Wing-length, Body-width, and Wing-width.

The features and predictors are given in a data frame to retrieve the data in a table format.

```
size = sample_df.shape[0] #size of sample
features = 4  #number of features
predictors = 3  #number of types(predictors)
```

Data-Normalizations:

The Dataset is later normalized, normalization of a dataset is necessary because the scales for different features are wildly different, this can have a knock-on effect on your ability to learn. Hence the different scales are brought down to a common scale.

```
#Normalization of Data
for k in range(n):
    X[:, k] = (X[:, k] - X[:,k].mean())

print(X)
```

Splitting Test and Training Data:

After normalizing the dataset is split into train and test of 2/3 and 1/3 respectively. The dataset is split into X_train,X_test, y_train and y_test.

```
y = sample_df['Type'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 11)
```

Sigmoid Function:

The equation to calculate logistic regression is given above in the report. The formula is applied to function Sigmoid. It returns the value which is used to find the accuracy of the given data. The predicted value can be anywhere between negative or positive infinity. The sigmoid function is used to determine the output variable as 0 or 1.

```
#Sigmoid fuction
def sigmoid(z):
    f= 1.0 / (1 + np.exp(-z))
    return f
```

LogisticRegression Function:

The logistic regression function is used to minimize the values as much as possible to reduce the cost value. Logistic function implements cost function and gradient descent.

regCostFunction, regGradient Function:

A cost function is used to minimize and Gradient descent is a method for finding the minimum of a function of multiple variables. Gradient descent to minimize the cost function. Gradient descent is an optimization algorithm and its responsibility are to find the minimum cost value. If the cost value is low, it is necessary to check if it can be minimized further.

```
#Optimal beta
def logisticRegression(X, y, beta):
  result = minimize(fun = regCostFunction, x0 = beta, args = (X, y),
                    method = 'TNC', jac = regGradient) #minimize function used using scipy

  return result.x
```

One vs all function:

The idea to use one vs all algorithm is to convert multiple class into two class. Where we set labels as 0 or 1. The index is iterated and the values are stored accordingly. The algorithm has an advantage of interpretability. Since each class has only one classifier it is possible to gain knowledge by inspecting its corresponding classifier. One is all algorithm implements the logistic regression function.

```
as_beta = np.zeros((k, n + 1))

#One vs all
i = 0
for j in y:
    #set the labels in 0 and 1
    temp_y = np.array(y_train == j, dtype = int)
    optimalBeta= logisticRegression(X_train, temp_y, np.zeros((n + 1,1)))
    as_beta[i] = optimalBeta
    i += 1
```

Accuracy:

10-cross validation is applied to the algorithm to determine the average accuracy of the algorithm.

```
k=0
k_fold=10
total_acc=0
for k in range(0,k_fold): #10  times procedure
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 11)
  P = sigmoid(X_train.dot(as_beta.T))
  p = [Type[np.argmax(P[i, :])] for i in range(X_train.shape[0])]
  accuracy=np.mean(p==y_train)    #accuracy
  total_acc= total_acc+accuracy
  k=k+1
print("10 cross validation : accuracy",(total_acc/k)*100)          #average accuracy
```

The accuracy is number of corrected predicted values divided by length of dataset. Corrected predicted values are determined by passing X_train dataset into the sigmoid function as an argument and it is compared with y_train dataset. (optional)

```
#Accuracy Calculation from Scratch

P = sigmoid(X_train.dot(all_theta.T))
p = [y[np.argmax(P[i, :])] for i in range(X_train.shape[0])]
accuracy=np.mean(p==y_train)
#res=accuracy*100
print("Model Accuracy from Scratch ", accuracy * 100 , '%')
```

To determine the efficiency of our algorithm. We have also determined the accuracy of the dataset using scikit-Learn.(optional)

```
accuracy_score(y_train, p) * 100
```

CONCLUSION:

Logistic regression is simple and will perform better than support vector machine and neural networks. It is not as complex as decision tree and efficiency of logistic regression is higher than decision tree. Logistic regression should be the first algorithm to learn about data mining because it helps us understand the pipeline of modelling. Logistic regression is a very powerful algorithm even for very complex problems. Logistic regression can achieve 95 % accuracy with right set of features.

APPENDIX:

Python Code		Documentation	
Shruthi-18230295	Niranjana-18230368	Shruthi-18230295	Niranjana-18230368
Data-Normalizations	Data-Loading, Assigning and attributes	Multi-nominal regression	Implementation and algorithm details
Sigmoid Function	Splitting Test and Training Data	Cost function	Sigmoid function
regCostFunction	logisticRegression Function	Algorithm Design-Data-Normalizations	Algorithm Design-Data-Loading, Assigning and attributes
regGradient Function	One vs all function	Algorithm Design-Sigmoid Function	Algorithm Design-Splitting Test and Training Data
10-cross/Accuracy Calculation	10-cross/Accuracy Calculation	Algorithm Design-regCostFunction, regGradient Function	Algorithm Design-logisticRegression Function
		Algorithm Design-Accuracy/10-cross	Algorithm Design- One vs all function
		Conclusion	Algorithm Design-Accuracy / 10-cross

Python Code:

```
#Packages to Import
import pandas as pd
from google.colab import files
import numpy as np
from sklearn.cross_validation import train_test_split
from scipy import optimize as op
from scipy.optimize import minimize
from sklearn.metrics import accuracy_score
import pandas as pd
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sb
import io
import itertools
import numpy as np
```

```
#Uploading File
uploaded = files.upload()

sample_df = pd.read_csv(io.StringIO(uploaded['owls.csv'].decode('utf-8')),header=None)
print(sample_df)

#Adding Attributes to the dataset
sample_df.columns=["Body-length","Wing-length","Body-width","Wing-width","Type"]
sample_df

#optional-Scatter plot for Boady length and wing length
OwlLength = sb.FacetGrid(sample_df, hue="Type", size=6).map(plt.scatter, "Body-length", "Wing-length")
plt.legend(loc='upper left');

#optional-Scatter plot for Boady width and wing width
OwlLength = sb.FacetGrid(sample_df, hue="Type", size=6).map(plt.scatter, "Body-width", "Wing-width")
plt.legend(loc='upper left');

Type = ['BarnOwl', 'LongEaredOwl', 'SnowyOwl']
size = sample_df.shape[0] #size of sample
features = 4  #number of features
predictors = 3  #number of types(predictors)

X = np.ones((size,features + 1))
y = np.array((size,1))
X[:,1] = sample_df['Body-length'].values
X[:,2] = sample_df['Wing-length'].values
X[:,3] = sample_df['Body-width'].values
X[:,4] = sample_df['Wing-width'].values
print(X)
```

```
#Normalization of Data
for k in range(features):
    X[:, k] = (X[:, k] - X[:,k].mean())
print(X)

#Splitting Test and Training data
y = sample_df['Type'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 11)

#Sigmoid fuction
def sigmoid(z):
    f= 1.0 / (1 + np.exp(-z))
    return f

s#Regularized cost function
def regCostFunction(beta, X, y,lr = 0.1):
    m = len(y)
    h = sigmoid(X.dot(beta))
    reg = (lr/(2 * m)) * np.sum(beta**2)

    return (1 / m) * (-y.T.dot(np.log(h)) - (1 - y).T.dot(np.log(1 - h))) + reg
#Regularized gradient function
def regGradient(beta, X, y, lr = 0.1):
    m, n = X.shape
    beta = beta.reshape((n, 1))
    y = y.reshape((m, 1))
    h = sigmoid(X.dot(beta))
    reg = lr * beta /m

    return ((1 / m) * X.T.dot(h - y)) + reg
```

```
#Optimal beta
def logisticRegression(X, y, beta):
  result = minimize(fun = regCostFunction, x0 = beta, args = (X, y),
                    method = 'TNC', jac = regGradient) #minimize function used using scipy

  return result.x

  as_beta = np.zeros((k, features + 1))

#One vs all
i = 0
for j in Type:
  temp_y = np.array(y_train == j, dtype = int)
  optimalBeta= logisticRegression(X_train, temp_y, np.zeros((features + 1,1)))
  as_beta[i] = optimalBeta
  i += 1

k=0
k_fold=10
total_acc=0
for k in range(0,k_fold): #10  times procedure
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 11)
  P = sigmoid(X_train.dot(as_beta.T))
  p = [Type[np.argmax(P[i, :])] for i in range(X_train.shape[0])]
  accuracy=np.mean(p==y_train)    #accuracy
  total_acc= total_acc+accuracy
  k=k+1
print("10 cross validation : accuracy",(total_acc/k)*100)          #average accuracy

#Accuracy Calculation from Scratch

P = sigmoid(X_train.dot(as_beta.T))
```

```
p = [Type[np.argmax(P[i, :])] for i in range(X_train.shape[0])]
accuracy=np.mean(p==y_train)
#res=accuracy*100
print("Model Accuracy from Scratch ", accuracy * 100 , '%')
print("Model Accuracy using scikit-learn ", accuracy_score(y_train, p) * 100 , '%')
```

REFERENCES

[1] https://beckernick.github.io/logistic-regression-from-scratch/

[2] https://cran.r-project.org/web/packages/sgmcmc/vignettes/logisticRegression.html
https://gluon.mxnet.io/chapter02_supervised-learning/softmax-regression-scratch.html

[3] https://stats.stackexchange.com/questions/111388/derivation-of-regularized-linear-regression-cost-function-per-coursera-machine-l

[4] https://www.geeksforgeeks.org/understanding-logistic-regression/

[5] https://datascience.stackexchange.com/questions/8657/trying-to-understand-logistic-regression-implementation

[6] http://crawles.com/lr-scratch/

[7] https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220

[8] https://medium.com/@rrfd/what-is-a-cost-function-gradient-descent-examples-with-python-16273460d634

[9] https://medium.com/we-are-orb/multivariate-linear-regression-in-python-without-scikit-learn-7091b1d45905

[10] https://www.pugetsystems.com/labs/hpc/Machine-Learning-and-Data-Science-Multinomial-Multiclass-Logistic-Regression-1007/

[11] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

[12] https://lmfit.github.io/lmfit-py/fitting.html

[13] https://scikit-learn.org/stable/modules/multiclass.html

[14] https://stackoverflow.com/questions/13623113/can-someone-explain-to-me-the-difference-between-a-cost-function-and-the-gradien