Assignment – 2

Swapnil Siddharth Course: 1CSD1 Student ID: 21230014

Algorithm: Logistic Regression

Logistic Regression is a classification technique which is used to predict distinct set of classes based on given observations. As linear regression technique produces continuous output values, logistic regression transforms those output values into a probability value using a sigmoid function. These probabilistic values can then further be represented into two or more distinct classes.

Most machine learning algorithms pertains to certain assumptions, Naive Bayes makes assumptions of conditional independence of features, K-NN makes assumptions that the of the query point resembles the properties neighborhood points. The main assumption we make in the logistic regression algorithm is that our classes are almost linearly separable or in some cases perfectly separable. This means that if there are two class points, we can draw a hyperplane which can separate the positive class from negative class.

Let us suppose that π is the decision surface plane which best separates the two class points, X is the feature vector, W is normal (which can be also called as weights) to the plane and B is the in the intercept. Then, $W^T X + B = 0$ is the equation of the plane, where W and X are vectors and B is a scalar.

Then we can pass this equation to the sigmoid function, which intakes any real valued number and provides an output between the range of 0 and 1. Hence on applying the sigmoid function, we get: $Y = \sigma(W^T X + B = 0)$, where Y is the prediction or the probability of event occurrence. If the probability is greater than 0.5, we classify it as 1, if probability is less than 0.5, we classify it as 0.

Our task in model building of Logistic Regression is to find the best W and B which corresponds to the plane such that can best separate positive points from negative points, and hence it can help us in predicting the class labels more accurately.

Design Schemes:

Linear Model

Logistic Regression is based on the fact that the target variable is categorical in nature and has only two possible classes. And using this analogy, it can apply probabilistic approach to find the occurrence of any event. We initially calculate the linear combination of dependent and independent variables, which is basically the linear regression model, which will provide us continuous output values. On that model we apply the sigmoid function which will give us a probability.

The decision surface in Logistic Regression is a hyperplane or line. And hence any point which is on the direction of the normal to the plane will be positive and the points which is opposite direction of the normal will be negative. Finding the right plane is the main task in building model such that it maximizes sum of correctly classified points(positive) and misclassified points(negative). But here, outliers can highly impact the plane, and consequently the sum of sum of correctly classified points(positive) and misclassified points(negative). Thus, we can do squashing using sigmoid function, where the maximum value is 1 and minimum is 0.

Sigmoid Function

The Sigmoid function is creating an S-shaped curve and is often called as squashing function as it can intake any real valued number and its output range is between 0 and 1. The advantage of this function is that it can input a very large positive number or a very large negative number, but the function will give output in between 0 and 1. If the curve tends towards positive infinity, the predicted value will be close to 1, consequently if the curve tends towards negative infinity, the predicted value will be almost 0.

Since outliers can highly impact the decision surface(plane) and our summation of correctly classified points(positive) and misclassified points(negative) in our linear model, performing squashing using sigmoid function can solve this issue, where the maximum value is 1 and minimum is 0. If the value is small, the curve will grow almost linearly and beyond 1 it will taper off and similarly beyond 0, it will taper off the other side.

The advantage of sigmoid function is that it has a probabilistic interpretation and can be differentiable. From probabilistic point of view, if a point is on very far the direction normal to the plane from the plane, the probability that it's a positive point reaches close to 1 and if very far from opposite to the direction of plane it will be close to 0.

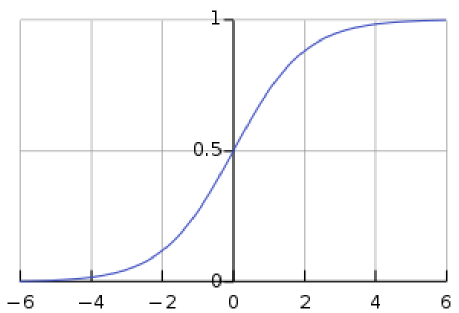

Figure 1: Sigmoid Function logistic curve

Source: Wikipedia

Cost Function

Cost function is basically to represent the errors present in our model building. It can help us to calculate how correctly our model is predicting the labels as compared to the actual class labels in the train dataset. If the cost function is more, the accuracy of predicting the actual class will reduce, consequently, if the cost function is less, the accuracy of predicting the actual class will increase. In Logistic Regression, we use cross-entropy cost function which is called log loss. The advantage of using cross-entropy is that it uses the concept of monotonic functions to create the cost functions.

A function $f(x)$ is said to be monotonic function such that if x increases $f(x)$ also increases and vice versa. Thus, this helps us in creating smooth cost function graphs which helps us in calculating the gradient and reducing the cost.

Thus, the cost function for logistic regression can be represented as:

$$cost = \frac{1}{n}\sum_{i=1}^{n} Yi \ \cdot \log\left(p(y^i)\right) + (1 - y^i) \cdot \log\,(1 - p(y^i))$$

Here, y^i is the actual class label and $\log\left(p(y^i)\right)$ is the probability of that class label.

Gradient Descent

Gradient descent is an optimization algorithm which helps us in finding the global or local minima of the function. It is inherently done to reduce the cost function. This optimization technique helps the model to learn the gradient in other words the direction which model should take to reduce errors. On plotting the graph of cost function, the main task of gradient descent is the calculate the weight and bias such that, as it starts from the top of curve with every step iteration it moves down in the direction which minimizes the cost function straight away. This is done iteratively such that we can reach the bottom of the graph which becomes our local minima. Here, the learning rate decides the step size the gradient should make in each iteration to reach the minima. The learning rate, which is low, where we are recalculating the minima very frequently, is much preferred as it can more precisely move in the direction of negative gradient. The whole process is integral, and the advantage of sigmoid function used in Logistic Regression is that its derivative is easy to calculate. Hence, we can build an equation for cost function derivate as:

$$W = x(p - a)$$

where, W = derivative of cost with respect to weights

p = prediction by the model

a = actual class label (0 or 1)

x = feature vector

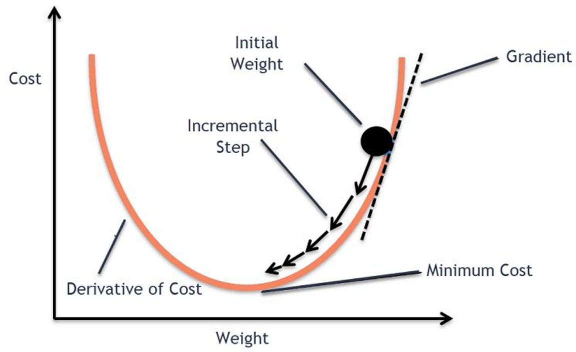

Figure 2: Gradient Descent

Source: Clairvoyant

Tests and Results:

Using the above-mentioned designed scheme, an implementation has been done to build the logistic regression classifier. The input file(wildfires) is a text file, upon which data cleaning and pre-processing has been performed such that the unneeded spaces have been removed and proper header to each feature column has been provided. Then using the Label Encoder provided with preprocessing under scikit-learn package has been used to turn the categorical value of yes and no for fire into 1 and 0. The training data has been randomly shuffled and divided dataset such as two-third of dataset is used for training and one-third is used for testing. The training data is then fit into the logistic regression model created as well as the built-in logistic regression classifier provided by scikit-learn package. The predictions of class labels are done for each model iteratively ten times and the accuracy score are calculated in each iteration. Afterwards, a mean accuracy score is calculated for implemented model as well as the reference model. Upon training the implementation of the logistic regression model on the train dataset and using the model to predict the class labels, we get an accuracy score of 72.35% as compared to 86.61% of the reference logistic regression classifier of scikit-learn.

```
Iteration Number: 1
Implemented Logistic Regression Classifier model accuracy: 69.11764705882352
Scikit-Learn Logistic Regression Classifier model accuracy: 88.23529411764706

Iteration Number: 2
Implemented Logistic Regression Classifier model accuracy: 86.76470588235294
Scikit-Learn Logistic Regression Classifier model accuracy: 88.23529411764706

Iteration Number: 3
Implemented Logistic Regression Classifier model accuracy: 63.23529411764706
Scikit-Learn Logistic Regression Classifier model accuracy: 83.82352941176471

Iteration Number: 4
Implemented Logistic Regression Classifier model accuracy: 67.64705882352942
Scikit-Learn Logistic Regression Classifier model accuracy: 85.29411764705883

Iteration Number: 5
Implemented Logistic Regression Classifier model accuracy: 64.70588235294117
Scikit-Learn Logistic Regression Classifier model accuracy: 86.76470588235294

Iteration Number: 6
Implemented Logistic Regression Classifier model accuracy: 75.0
Scikit-Learn Logistic Regression Classifier model accuracy: 85.29411764705883

Iteration Number: 7
Implemented Logistic Regression Classifier model accuracy: 79.41176470588235
Scikit-Learn Logistic Regression Classifier model accuracy: 91.17647058823529

Iteration Number: 8
Implemented Logistic Regression Classifier model accuracy: 63.23529411764706
Scikit-Learn Logistic Regression Classifier model accuracy: 89.70588235294117

Iteration Number: 9
Implemented Logistic Regression Classifier model accuracy: 72.05882352941177
Scikit-Learn Logistic Regression Classifier model accuracy: 85.29411764705883

Iteration Number: 10
Implemented Logistic Regression Classifier model accuracy: 82.35294117647058
Scikit-Learn Logistic Regression Classifier model accuracy: 82.35294117647058

Mean Accuracy of Implemented Logistic Regression Classifier model accuracy:  72.35294117647058
Mean Accuracy of Scikit-Learn Logistic Regression Classifier model accuracy:  86.61764705882352
```

Figure 3: Accuracy Results

Conclusions and observations:

After training the implemented logistic regression model on the train dataset and using the model to predict the class labels, we get an accuracy score of 72.35% as compared to 86.61% of the reference logistic regression classifier of scikit-learn. During initially training and testing of the model, we observed that the learning rate if increased and the number of iterations decreased, we are getting a lower accuracy score. Thus, upon experimenting with the learning rate and number of iterations, we find the ideal values of learning rate and number of iterations to be 0.0001 and 10000 respectively. This gives us a fairly good accuracy performance which is slightly less than the accuracy of the model trained with scikit-learn logistic regression classifier. An output file has also been created which stores the output predicted and actual values in a csv file.

References:

- **How Does the Gradient Descent Algorithm Work in Machine Learning? [https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/]**
- **Sigmoid function [https://en.wikipedia.org/wiki/Sigmoid_function]**
- **Scikit-Learn GitHub [https://github.com/scikit-learn/scikit learn/blob/main/sklearn/linear_model/_logistic.py]**
- **Logistic Regression [https://en.wikipedia.org/wiki/Logistic_regression]**
- **Cross entropy [https://en.wikipedia.org/wiki/Cross_entropy]**
- **Understanding Logistic Regression in Python [https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python]**

Appendix:

```python
#Importing required libraries for the assignment code

import pandas as pd #for data-preprocessing
import numpy as np #for creating nd-arrays and matrix multiplications
from sklearn.model_selection import train_test_split #for splitting dataset into test and train data
from sklearn.linear_model import LogisticRegression #for using Logistic Regression classification and compare it's accuracy with bulid classifier
from sklearn import metrics #for accuracy calculations of
from sklearn import preprocessing #for importing LabelEncoder

wildfires_data = open('wildfires.txt', 'r')


#Creating a function which removes any uneeded spaces in the text dataset
def remove_spaces(input_file):
    elements = []
    for line in input_file:
        element = []
        element = line.split()
        elements.append(element)
    return elements

#Creating the required column names of the dataset
column_names = ['fire', 'year', 'temp', 'humidity', 'rainfall', 'drought_code', 'buildup_index', 'day', 'month', 'wind_speed']

#Applying the function to remove spaces and creating the dataframe using pandas and giving the appropiate column names
wildfires_df = pd.DataFrame(remove_spaces(wildfires_data), columns = column_names)

#Dropping the first row of the dataframe as it contains column names
wildfires_df = wildfires_df.iloc[1:]

#Creating feature columns
features_columns = ['year', 'temp', 'humidity', 'rainfall', 'drought_code', 'buildup_index', 'day', 'month', 'wind_speed']

#creating labelEncoder
label_encoder = preprocessing.LabelEncoder()

#Label encoding the fire column
wildfires_df['fire'] = label_encoder.fit_transform(wildfires_df['fire'])
#print(wildfires_df)

#Creating X and y dataframes with required features and classes from wildfires dataset
X = wildfires_df[features_columns]
y = wildfires_df['fire']

#Creating Logistic Regression Classifier
class LogisticRegressionClassifier:

    #Creating init method and initializing the values 0.001 and 1000 to learning rate and number of iterations for gradient descent respectively
    def __init__(self, learning_rate=0.001, number_of_iterations=1000):
        self.learning_rate = learning_rate
        self.number_of_iterations = number_of_iterations
        #Creating weights and bias as None initially, which laters need to be calculated
        self.weights = None
        self.bias = None

    #Creating the sigmoid function
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    #Creating method to calculate gradient descent and updating weights and bias
    def gradient_descent(self, n_sample, p_score, actual, Z):

        #Calculating the derivative of wegights and bias
        dw = (1 / n_sample) * np.dot(Z.T, (p_score - actual))
```

```
        db = (1 / n_sample) * np.sum(p_score - actual)

        #Adjusting the weights and bias with respect to learing rate
        self.weights = self.weights - self.learning_rate * dw
        self.bias = self.bias - self.learning_rate * db

    #Creating the fit method which will take training dataset and labels as parameters
    def fit(self, train_data, target_labels):

        #Initializing the parameters
        number_of_samples = train_data.shape[0]
        number_of_features = train_data.shape[1]
        self.weights = np.zeros(number_of_features)
        self.bias = 0

        #Running iterations for building linear model to fit into sigmoid functtion and updating weights and bias using gradient descent function
        for _ in range(self.number_of_iterations):

            #Creating a linear model with y and dot product of weights and training features data adding bias
            linear_model = np.dot(train_data, self.weights) + self.bias

            #Applying the build sigmoid function to the linear model to find the probilistic scores(approximations) of y
            probability_scores = self.sigmoid(linear_model)

            #Calling Gradient Descent function
            self.gradient_descent(number_of_samples, probability_scores, target_labels, train_data)

    def predict_class(self, X):

        #Creating a linear model with y and dot product of weights and x adding bias
        linear_model = np.dot(X, self.weights) + self.bias

        #Applying the build sigmoid function to the linear model to find the approximation of y
        probability_scores = self.sigmoid(linear_model)

        #Using list comprehension to predict classes(values > 0.5 as 1 and values < 0.5 as 0)
        predicted_classes = [1 if i > 0.5 else 0 for i in probability_scores]
        return np.array(predicted_classes)



#Defining the accuracy function to calculate the accuracy of the build model
def accuracy_score(actual_class, predicted_class):
        accuracy_score = np.sum(actual_class== predicted_class) / len(actual_class)
        return accuracy_score


#Building list to store accuracy for 10 iterations and calulating mean accuracy score
build_model_accuracy_list = []
scikitlearn_model_accuracy_list = []

for i in range(10):

    print("Iteration Number:",i+1)

    #Splitting the dataset into train and test datasets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, shuffle = True)

    #Converting the train and test datasets into numpy nd-array
    X_train, X_test, y_train, y_test = X_train.values , X_test.values , y_train.values , y_test.values

    #Converting the nd-array of test and train dataset into float values in order to facilitate matrix multiplications
    X_train, X_test, y_train, y_test = X_train.astype(float) , X_test.astype(float)
```

```
        , y_train.astype(float) , y_test.astype(float)

        #Creating a classifier object using the implemented(built) Logistic Regression classifier and setting parameters 0.0001 and 10000 for learning rate and number of iterations
        logistic_regression_build_model = LogisticRegressionClassifier(learning_rate=0.0001, number_of_iterations=10000)

        #Calling the fit function to train the model with the train datasets
        logistic_regression_build_model.fit(X_train, y_train)

        #Predicting the class on test dataset using the model
        build_predictions = logistic_regression_build_model.predict_class(X_test)

        #Calculating the accuracy score with accuracy function
        build_model_accuracy = (accuracy_score(y_test, build_predictions))*100

        #Printing the accuracy in each iteration
        print("Implemented Logistic Regression Classifier model accuracy:", build_model_accuracy)

        #Appending the accuracy of each iteration in the respective list
        build_model_accuracy_list.append(build_model_accuracy)


        #Creating Logistic Regression classifier object using Sk-learn Logistic Regression classifier
        scikitlearn_logistic_regression_model = LogisticRegression(solver='lbfgs', max_iter=1000)

        #Training Logistic Regression Classifier
        logisticregression_clf = scikitlearn_logistic_regression_model.fit(X_train, y_train)

        #Predicting the response from test dataset
        scikitlearn_model_predictions = logisticregression_clf.predict(X_test)

        scikitlearn_model_accuracy =  (metrics.accuracy_score(y_test, scikitlearn_model_predictions))*100
        print("Scikit-Learn Logistic Regression Classifier model accuracy:", scikitlearn_model_accuracy,"\n")
        scikitlearn_model_accuracy_list.append(scikitlearn_model_accuracy)


    print("Mean Accuracy of Implemented Logistic Regression Classifier model accuracy: ", np.mean(build_model_accuracy_list))

    print("Mean Accuracy of Scikit-Learn Logistic Regression Classifier model accuracy: ", np.mean(scikitlearn_model_accuracy_list))

    #Creating dictionary of actual and predicted labels of model and storing it in an output file
    actual_labels_and_predicted_labels = pd.DataFrame({'Actual Label': y_test, 'Predicted Label': build_predictions})
    actual_labels_and_predicted_labels.to_csv("label_outputs.csv", index = False)
```