Machine Learning Assignment 2

Implementing Logistic Regression from Scratch

Kollapudi Nagendra Dheeraj

Student Id: 20231272

Class: 1CSD

As part of the Machine Learning Assignment, the algorithm that I implemented is Logistic Regression Classifier from scratch using the Python Programming Language over the beer dataset. Along with the use of some packages for csv data handling and computation of data – some functions in SciKit Learn, NumPy and Pandas.

Along with that for the Reference Implementation I opted for SciKit Learn's Logistic Regression from sklearn. linear_model module.

Description of the Algorithm:

Logistic Regression is another type of Supervised Learning Classification algorithm that uses a complex cost function to predict the class of unseen feature values. This cost function is a Sigmoid Function denoted as: $\frac{1}{1+e^{-x}}$ along with the representation of hypothesis for single feature as: $h_\theta(x) = g(\beta_0 + \beta_1 * X_1)$ thus the resulting hypothesis with the cost function can be shown as: $h_\theta(x) = \frac{1}{1+e^{-\beta_0+\beta_1 * X_1}}$

Now as with Linear Regression, this Logistic Regression also deals with the Parameters and their weights and to compute the best class for a test feature so we need to apply an Optimization Algorithm to find the minimum cost for our hypothesis function with cost variable in it.

So we also have to consider the Gradient Descent: $\theta^{new} = \theta^{old} - \alpha \frac{1}{m} \sum_{i=1}^{m} x_j^i \, (h_\theta(x^i) - y^i)$

The above highlighted procedure is for the basic Binary Classification of the Logistic Regression.

But the dataset provided is clearly for the Multiclass Logistic Regression. The procedure that I used for implementing Multiclass LR and also that is highlighted in my ipynb file is as follows:

1. Loading the Dataset: After importing all the required Python Packages that is Pandas, NumPy and few sklearn functions for scaling, splitting the test and train dataset and accuracy score. I imported the dataset added column values to them for easy processing.

2. Extract Features and Labels for Processing: Separating out the Features (X values based on which computation is being done) and labels (Y values which my model should be able to predict)

3. Perform Feature Scaling over X values: Feature scaling helps in brining data values that are in extremely different ranges or units into standard format. Bringing all the measurements into a similar scale helps the classifier perform better, and I created my

own implementation for Normalization that behind the scenes uses, Mean, Standard Dev values or in-fact standard Normalization and not 0/1 Normalization.

4. Logistic Regression Hypothesis (Sigmoid Function): As mentioned above, we are writing sigmoid function over here to distinguish in a traditional fashion a binary classification. We used NumPy's exponent function as it is vectorized and automatically works on a list of values instead of for loop.

5. Logistic Regression Gradient Descent: As written above, I implemented the formula into the python code that at first computes the hypothesis, and then uses that value to calculate the gradient value and then multiples it with the learning rate and this entire value is reduced with the older value. Using epochs to slowly get to the minimum value.

Steps 1 – 5 are common in LR either it be Binary or Multiclass. The Below 2 Steps are different when it comes to Binary vs Multiclass.

6. Logistic Regression Fit: In this part we are interested in developing the theta value matrix of 8 columns as columns of interest are 8 replicated across 3 rows corresponding to 3 classes. This is achieved via this procedure:

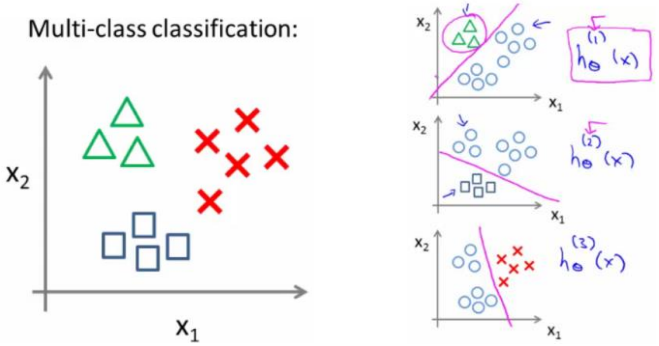
    a. As this is a Multiclass, we kind of implement it as Binary but over multiple iterations. This strategy is called as One vs Rest Methodology. It goes as:

    b. First, I computed the unique classes from my y_train and then created an empty array of 3 * 8 to accommodate values.

    c. I used a for loop to iterate over each unique class.

        i. Then created new y_label but in form of 1, 0 where 1 if y_label is same as the current iterated value and 0 if otherwise.

        ii. Passed all the values to calculate the theta values over this Binary classification. Append the returned value list to the main list of empty thetas.



    d. Return the theta values.

7. Logistic Regression Predict: This is another critical part as in the traditional approach we could have taken a simple cut-off of 0.5 to consider which class did the unseen data belonged to, but in the multiclass scenario this entire thing is different. So, we followed this procedure:

    a. Calculate the hypothesis but in this scenario, theta will not be 0 but the pre-computed values.

b. Find the hypothesis which would compute in this style $[\theta_1^1 \quad \theta_2^1 \quad \theta_3^1]$ that has the highest value and return the index of it.

c. Map the index returned to the class and this is the predicted class by the system.

Design Decisions:

Some of the design decisions that were taken as part of the Algorithm Development are:

1. Using NumPy for array calculation as far as possible as it deals with vectorized calculation, otherwise I might had to end up creating empty arrays, use for loop, append the results, which is a bit tedious and more error prone and at times hard to understand.

2. Use of sklearn function for some intermediate processing like accuracy score, train_test_split etc., as the main focus in this assignment is about the LR algorithm development from scratch so left these manually implementable activities to already existing function for easy understanding.

3. Relying on NumPy function for predicting the class over the unseen data as it enabled me to develop the algorithm in a smaller number of lines that is also easy to understand and modifiable for further changes.

Tests & Results: In my previous Assignment I used ID3 which gave test accuracy of 82.7% and Random Forest Classifier which gave test accuracy of 84%, so comparing these Logistic Regression for both mine and sklearn was around [95 - 97]%. So, for this dataset it can be easily concluded that Logistic Regression gives pretty superior results.

As this does not compare my implementation with sklearn's in this Assignment, I performed 4 different comparative tests – ROC Curve, Accuracy Comparison, Confusion Matrix, Classification Report against my implementation and SkLearn's LR.

Here I would highlight why these tests and what were the results:

1. Accuracy vs Iteration: The simplest of all, Accuracy score for each case is considered for all the 10 iterations and plotted in the graph. In most of the cases, my implementation was able to predict slight better than sklearn. Sklearn accuracy was almost standard while mine was fluctuating.

   Results: Both of them gave an average that is comparable without huge deviation, but for some train test split my implementation gave certain low accuracies while sklearn's maintained in the standard in its fluctuations.

2. Receiver Operating Characteristic Curve: The Area Under the ROC Curve would help us highlight the probability of each class as per the classifier prediction. In which we use the True Positives and False Positives, and we expect the curve to lean towards the top right corner for optimal performance. Along with that provided is the Area Under ROC for better understanding of the classifier performance to predict a particular class.

   Results: Both the ML Package and self-Implementation were leaning against the left corner and especially for the case of stout the correct classification probability was far higher in both cases, followed by ale and lager.

3. Confusion Matrix: Another straightforward metric to understand the performance difference is the confusion matrix that can easily visualize the times when classifier was

able to correctly classify the differences. This provided a clear retrospect on why there is a performance difference, so from this it was noticeable that when true label – lager, sklearn classifier was misidentifying it as ale for more cases than my implementation. So, the performance difference can be identified from this inconsistency. Also, other metrics were computed for better analysis of the data – Precision, Recall and F1-Score and as specified earlier due to one small mishap Precision scores are different but rest Recall and F1-Scores are similar.

Results: Both of these implementations worked well on the data, as seen from the Recall and F1-Score being similar. But due to the misclassifications of ale and lager, Precision and Accuracy went a bit down.

4. Classification Report: In here, it is the same thing that I was doing above Precision, Recall and F1-Score but in here instead of entire classification predictions we are taking for specific classes for better retrospection. As indicated above, the issue lied in the lager and ale conflict that reduced the accuracy index.

   Results: Both of these implementations worked well on the data, as seen from the Recall and F1-Score being similar. But due to the misclassifications of stout and lager, Precision and Accuracy went a bit down.

Conclusions & Observations:

Observations:

1. At par with the Sklearn's implementation, my implementation too was able to significantly classify the data and predict the appropriate label class. But still the accuracy is a bit off for sklearn due to lager-ale conflict.
2. Maybe sklearn has some parameter that it uses behind the scenes which helps it to maintain the classification. Epochs value is also another concern that might affect the accuracy.
3. From the Accuracy vs Iteration graph, the algo is heavily prone to train data split, as it is showing heavy fluctuations in a short range. While Sklearn was less fluctuant.
4. Under other metrics, stout classification was optimal while ale and lager classification were a bit varying. If the test case had a stout there is higher chance of it classifying the data as lager is higher.
5. Recall and F1-score was pretty nearby, it means the expected behavior doesn't vary but the Precision is a bit off, signifying that there are some cases where the prediction differences are occurring.

Conclusions:

1. Vectorized implementation helped save a lot of for iterations and helped the system execute operations faster. 10 iterations with 10000 epochs was executed in under 4s.
2. Even with straight forward approach for this dataset the classifier was able to be a par with the sklearn classification.
3. Logistic Regression is actually a bit for this dataset, as in cases of other classifiers their implementation and hypothesis being complex. LR is the one that fits the data perfectly as per the Occam's Razor – to go with the simplest one. This is a perfect fit.