Multiclass Classification Using K-Binary Logistic Regression Classifiers

Abdul Wahab, *Student, Bilkent University EEE*

***Abstract*—The aim of this report is to demonstrate the use of K-Binary Logistic Classifiers for K class classification as a valid and efficient technique compared to the relatively complex and computationally cumbersome Multiclass Logistic Regression techniques for K class classification such as the one implemented by MATLAB as mnrfit() in Statistics and Machine Learning Toolbox™. For testing purposes we used a dataset from UCI [1] data sets repository. The data set is described in detail in the report. Detailed formulation of the classification based on K-Binary Logistic Classifiers is provided. The developed method can be downloaded as a MATLAB toolbox.**

I. INTRODUCTION

Logistic Regression is a commonly known classification method. We will refer to multiple logistic regression (logistic regression with more than one prefictor variable dimension) to as simply logistic regression. Moreover, we will refer to binomial/binary i.e. having only **K=2** to as binary logistic regression. For more than two class logistic regression, multiclass logistic regression is the commonly used term and we shall do so as well. The objective is to validate the use of binary logistic regression based multiclass classification in place of conventional multiclass logistic regression techniques. Binary logistic regression is a fairly simple classification method. The logistic function is the basis for this classification algorithm. Logistic function provides with a probability estimate of certain outcome given an observation/data. This logistic function can be extended to multiple classes in a rather simple way.

II. LOGISTIC REGRESSION CLASSIFICATION

Although binary logistic regression is a well-defined and commonly used technique, multiclass logistic regression classification is relatively uncommon. The Newton-Raphson update equation for multiclass logistic regression is not easily available in the literature. Most of the ones available involve a much bigger weights matrix (bigger by factor of K as compared to binary logistic regression). The size of this weight matrix renders the multinomial logistic regression to be computationally cumbersome. This problem worsens with increase in the number of training instances and/or dimensions (i.e. the design matrix gets bigger) of the predictor variable. Hence, multiclass logistic regression is computationally expensive and therefore, infeasible.

A. MATLAB Multiclass Logistic Regression Function(mnrfit())

MATLAB provides mnrfit [2] function as part of the Statistics and Machine Learning Toolbox™. We used our dataset described earlier and used the first 16000 instances for training. The function returns the $\boldsymbol{\beta}$ coefficients matrix as is expected of a logistic regression classification system. The algorithm runs for around 3680 seconds before converging and returning the coefficients matrix. These algorithm running time statistics are for a Core-i5 3330 desktop PC equipped with 8GB of RAM and no other parallel program running. The accuracy achieved on the set of test data with 4000 last instances was around 76% while that achieved on the training data set of 16000 instances was 77%. For predicting the outcome, mnrval() function from MATLAB's Statistics and Machine Learning Toolbox™ was used.

B. Alternative Solution: Use K Binary Classifiers

Instead of using multiclass logistic regression we can use K Binary Classifiers for a K classification problem. This is similar to binary classification where there are two classes where assignment of a test input is dependent on a certain probability threshold which is computed using the logistic function given below:

$$\pi(\boldsymbol{X};\beta) = \frac{e^{\beta_0+\beta_1 X_1+\ldots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\ldots+\beta_p X_p}} = \frac{e^{\boldsymbol{\beta}^T \boldsymbol{X}}}{1+e^{\boldsymbol{\beta}^T \boldsymbol{X}}}$$

In the above formulation, $\boldsymbol{X}$ is a predictor variable with p dimensions.

The logistic function above models the probability of test input $\boldsymbol{X}$ belonging to the class of $\boldsymbol{\beta}$ vector. We will have 26 such $\boldsymbol{\beta}$ vectors, trained from dataset for each of the 26 Alphabet classes. Hence, for a single test input, its corresponding class is predicted by the index of the $\boldsymbol{\beta}$ yielding the highest probability value (logistic function value).

C. MATLAB implementation for K-Binary Classifiers

We used MATLAB as the platform for scripting and testing our algorithm for multiclass classification. The dataset described was used. The first 16000 instances were used for training.

The response data consists of alphabets as characters from 'A' to 'Z'. These response data alphabets were mapped to

integers. For instance, 'A' mapped to integer 1 'Z' mapped to integer 26. This was mainly done to make it compatible to our algorithm which will take class labels as integers so that it can cater to general classification tasks. This is done by **getMapping.m**..

For training each of the K Binary Classifiers, the algorithm maps the integer response value to 1 for if the Kth classifier is being trained else 0. i.e. if the $\boldsymbol{\beta}$ vector being trained belongs to class K, and the response value is K, the response value will become 1 else zero. This mapping is in line with the binary logistic classifier approach. We use **getY_blr.m** to achieve this.

The logistic regression model training stage involves finding out MLE (Maximum Likelihood Estimate) estimates for the coefficient $\boldsymbol{\beta}$ vector. This MLE is achieved using newton-Raphson method. Newton-Raphson method employs update coefficients, $\boldsymbol{\beta}$ vector update equation for each of the K $\boldsymbol{\beta}$ vectors. The update equation is as follows:

$$\beta^{\text{new}} = \beta^{\text{old}} + (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T(\boldsymbol{y} - \boldsymbol{\pi})$$

$\boldsymbol{\beta}^{\boldsymbol{old}}$ is initialized to zero. We use **getB.m** to get each of the **K** $\boldsymbol{\beta}$ vectors.

W is the following matrix:

$$\begin{bmatrix} \pi(x_1;\beta^{\text{old}})(1-\pi(x_1;\beta^{\text{old}})) & 0 & \dots & 0 \\ 0 & \pi(x_2;\beta^{\text{old}})(1-\pi(x_2;\beta^{\text{old}})) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \pi(x_n;\beta^{\text{old}})(1-\pi(x_n;\beta^{\text{old}})) \end{bmatrix}$$

W is computed using **getW.m**

X has a column of ones (accounting for intercept's coefficients in $\boldsymbol{\beta}$ vector):

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{bmatrix}$$

Π (pi) is the following matrix:

$$\boldsymbol{\pi} = \begin{bmatrix} \pi(x_1;\beta^{\text{old}}) \\ \pi(x_2;\beta^{\text{old}}) \\ \vdots \\ \pi(x_n;\beta^{\text{old}}) \end{bmatrix}$$

Individual **Π** terms are defined earlier. **getPi.m** is used to compute the **Π** matrix above.

y is the following single column matrix of response data:

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

We update $\boldsymbol{\beta}$ until convergence i.e. $\boldsymbol{\beta}$ vector does not change.

III. DATASET DESCRIPTION

The dataset consists of each black and white image as one of the 26 capital English alphabets. The dataset was obtained from UCI Machine Learning Repository. The character images were based on 20 different fonts and each of the images was randomly distorted to produce 20000 unique images. Each image was first converted into 16 primitive numerical attributes like statistical moments and edge counts. These attributes were scaled to fit integers ranging from 0 through 15.The dataset was divided into training data (16900 instances) and test data (3100 instances). The best accuracy obtained on the dataset using Holland-style adaptive classifier was a little over 80%.

Number of Instances: 20000 (16900 – training, 3100 – test)
Number of attributes: 17 (Label of letter and 16 numeric features)

1. Letter capital letter (26 values from A to Z)
2. x-box horizontal position of box
3. y-box vertical position of box
4. width width of box
5. height height of box
6. onpix total # on pixels
7. x-bar mean x of on pixels in box
8. y-bar mean y of on pixels in box
9. x2bar mean x variance
10. y2bar mean y variance
11. xybar mean x y correlation
12. x2ybr mean of x * x * y
13. xy2br mean of x * y * y
14. x-ege mean edge count left to right
15. xegvy correlation of x-ege with y
16. y-ege mean edge count bottom to top
17. yegvx correlation of y-ege with x

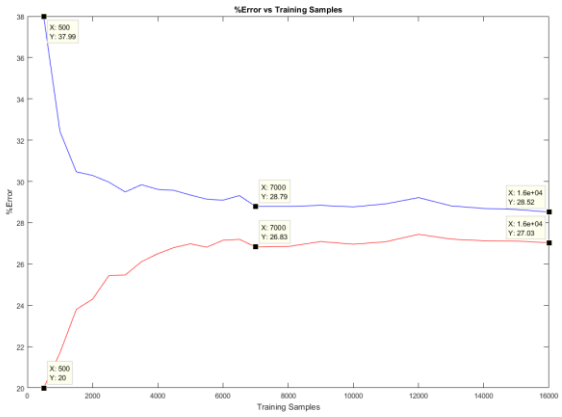
Since the data consists of all the integers in the range 0-16, we do not need to standardize/normalize data. However, it varies from case to case.

Class Distribution:

Alphabet	#Instances	Alphabet	#Instances
A	789	N	783
B	766	O	753
C	736	P	803
D	805	Q	783
E	768	R	758
F	775	S	748
G	773	T	796
H	734	U	813
I	755	V	764
J	747	W	752
K	739	X	787
L	761	Y	786
M	792	Z	734

IV. Results using K-Binary classifiers

The last 4000 instances of dataset were used for testing. The training set size was varied from first 1000 instances to first 16000 instances.



By employing our method of K-Binary Classifiers we achieve appreciable results. We are able to achieve an accuracy of about 71.5% (error is 28.5%) on test data and an accuracy of 73%(error is 27%) on training data using 16000 instances for training. Running time for this algorithm on corei5-3300 equipped with 4GB RAM is 300 seconds which is almost 12 times less than what is required my MATLAB for 76% accuracy on test data.

However, as can be seen K-Binary Classifiers converges rather quickly at 8000 training instances which requires much less time of around 100 seconds which is 36 times less than what is required by MATLAB although with a tradeoff in accuracy.

V. Conclusion

For K >2 classification, K-Binary Logistic Classifiers provide an effective solution for K class classification. Conventional Multiclass algorithms are relatively more complex to implement and computationally cumbersome. As can be seen in our case for an accuracy trade off from 76% (MATLAB mnrfit() function) to 71.5%(K-Binary Logistic Classifiers) we are able to reduce computational time by a factor of 12. Furthermore if we use a smaller training sample size of around 8000, the accuracy drops by about 0.3% (71.2% accuracy) we have time cost of around 60 seconds. Hence, we get a reduction of time training time by a factor of 60. Python implementation of the same algorithm will be made available in the next step.

VI. References

[1]. Letter Recognition Data Set . (n.d.). Retrieved March 04, 2018, from https://archive.ics.uci.edu/ml/datasets/letter recognition

[2]. Mnrfit. (n.d.). Retrieved March 04, 2018, from https://www.mathworks.com/help/stats/mnrfit.html

Appendix

The functions used for training the model are as below:

```
function [ B ] = MLR( X,Y )
B=zeros(26,17);
X=[ones(size(X,1),1),X];
for i=1:1:size(B,1)
    Y_blr=getY_blr(Y,i);
    B(i,:)=getB(B(i,:),X,Y_blr);
end
end
function [ Bn ] = getB( B,X,Y )
Bo=zeros(size(X,2),1);
for i=1:8
W=getW(X,Bo);
Pi=getPi(Bo,X);
Bn=Bo + pinv(X'*W*X)*X'*(Y-Pi);
Bo=Bn;
end
Bn=Bo;
end

function [ Pi ] = getPi(Bo,X)
Pi=[];
S=size(X,1);
W=zeros(S(1));
for i=1:1:S(1)
P=(exp(Bo'*X(i,:)'));
P=P/(1+P);
Pi=[Pi,P];
end
Pi=Pi';
end

function [ W,Ps ] = getW( X,Bo )
Ps=[];
S=size(X);
W=zeros(S(1));
for i=1:1:S(1)
num=(exp(Bo'*X(i,:)'));
denom=1+num;
P=num/denom;
RP=1-P;
size(W);
W(i,i)=P*RP;
end
end

function [ Y_blr ] = getY_blr(Y,i)
Y_blr=Y;
Y_blr(Y==i)=1;
Y_blr(Y~=i)=0;
end
```

The function for reconstruction the data is below

```
function [ Y ] = getPredMLR( B,X )
X=[ones(size(X,1),1),X];
Y=[];
for i=1:size(X,1)
    c=[];
        num=exp(B*X(i,:)');
        denom=1+num;
        Pi=num./denom;
        c=Pi;
        [m,i]=max(c);
        Y=[Y;i];
end
end
```

The Entire Project can be downloaded from the following link: https://goo.gl/o5YzyS

The MATLAB toolbox can be downloaded from the following link: https://www.mathworks.com/matlabcentral/fileexchange/65800-logistic-regression