Implementing a Binary Logistic Linear Classifier with Categorical and Real Valued Attributes in Python Programming Language

Alireza Zaeemzadeh
CAP 6676 - Machine Problem 2
University of Central Florida
Email: zaeemzadeh@knights.ucf.edu

Fall 2015

In this machine problem, a binary logistic linear classifier is implemented in python programming language.

Step 1: dataset includes both categorical and real valued attributes. A dataset is chosen from the University of California Irvine repository. This repository is available online at:

https://archive.ics.uci.edu/ml/datasets.html.

For this report, the dataset **Abalone** is employed to train and test the decision tree. The default task for this dataset is multi-class classification. The goal is to estimate the age of the abalone by predicting the number of the rings in the shell. There exist 29 classes, representing different number of rings. To reduce the problem into a binary decision problem, data points from two classes is employed to train and test the decision tree.

The candidate hypotheses should be chosen in such a way that produce enough number of data points. Also, it is worthwhile to mention that if two classes with similar attributes are chosen for the classification task, the performance of the classifier would decrease significantly.

To validate the performance of the tree, K-fold cross validation is used. In each fold, N/K data points are employed to test the data and the rest of data points are used to train the data. N is the total number of data points and the default value of K is set to 10.

Step 2: In the algorithm, the coefficients vector $\boldsymbol{w}$ is updated in each iteration by the following update rule:

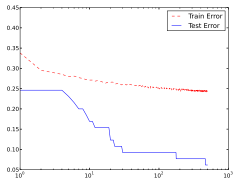
$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} + \alpha * \boldsymbol{g}^{(t)}(\boldsymbol{x}, \boldsymbol{w}^{(t)}), \tag{1}$$

where α is the learning rate and $\boldsymbol{g}^{(t)}(\boldsymbol{X}, \boldsymbol{w}^{(t)})$ is the gradient of the log-likelihood of the labels given the attributes $\boldsymbol{X}$ and the current coefficients $\boldsymbol{w}$. The gradient is calculated using the following formulation:

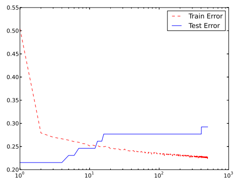
$$\boldsymbol{g}^{(t)}(\boldsymbol{X}, \boldsymbol{w}^{(t)}) = \sum_{m=1}^{M} \boldsymbol{x^m}(y^m - \mathbb{P}(y^m = 1|\boldsymbol{x^m}, \boldsymbol{w}^{(t)}))$$
$$\text{where} \quad \mathbb{P}(y = 1|\boldsymbol{x}, \boldsymbol{w}) = \frac{e^{w^T \boldsymbol{x}}}{1 + e^{w^T \boldsymbol{x}}}. \tag{2}$$

M is the size of the training data set. The coefficients are updated until a maximum number of iterations is reached. Then, to classify a test data, the conditional probability $\mathbb{P}(y = 1|\boldsymbol{x}, \boldsymbol{w})$ is calculated. If $\mathbb{P}(y = 1|\boldsymbol{x}, \boldsymbol{w}) > 0.5$, then $y = 1$.

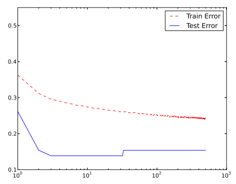
Figure 1 and Figure 2 illustrate the errors and log-likelihood over iterations, respectively. Also, it is easy to notice that over-fitting has occurred for the test case plotted in Figure 1b.



(a)

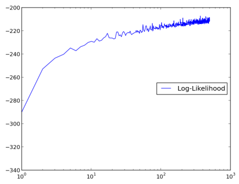


(b)

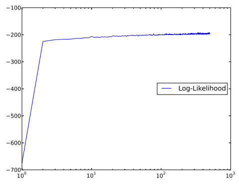


(c)

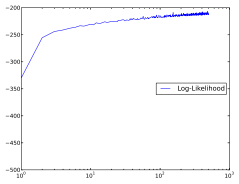
Figure 1: Test and Train Error vs Iteration.



(a)



(b)



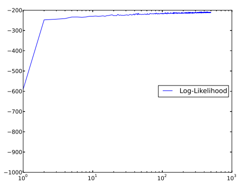
(c)

Figure 2: Log-Likelihood vs Iteration.

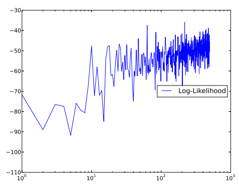
Step 3: In stochastic gradient ascent, the gradient is calculated using a subset of data, instead of the whole training data set. Figure 3 demonstrates the convergence of the algorithm for stochastic gradient ascent. As the size of data, which is employed to calculated the gradient, decreases the convergence of the algorithm deteriorates. However, the computational burden of the algorithm decreases significantly.

Step 4: To avoid over-fitting, we should employ prior knowledge about the coefficient vector $\boldsymbol{w}$. By assuming a zero-mean Gaussian distribution for the coefficients, the update rule is modified as:

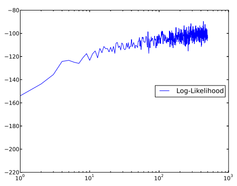
$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} + \alpha * (\boldsymbol{g}^{(t)}(\boldsymbol{x}, \boldsymbol{w}^{(t)}) - \lambda \boldsymbol{w}^{(t-1)}). \tag{3}$$



(a) regular gradient ascent



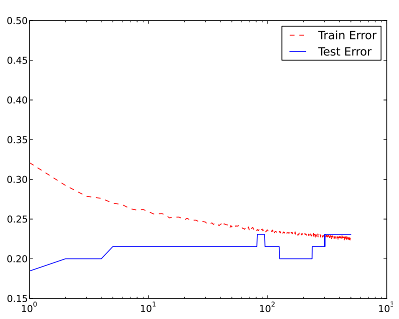
(b) stochastic gradient ascent with subset size 65



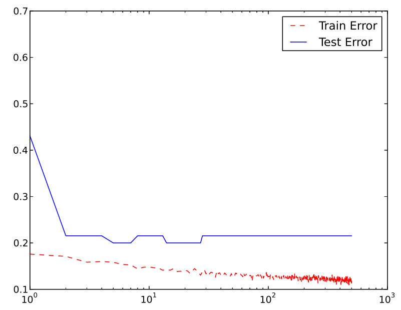
(c) stochastic gradient ascent with subset size 260

Figure 3: Log-Likelihood vs Iteration for Stochastic Gradient Ascent for a Training Data Set with Size $M = 585$.

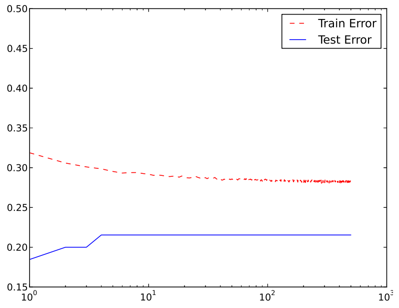
By using such update rule, the coefficients tend to be small. Figure 4 shows the performance of the regularized problem. It is easy to notice that the effect of the over-fitting has been reduced after assigning a non-zero value to the regularization parameter λ.



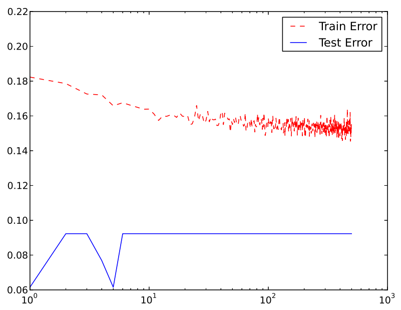
(a) regular gradient ascent with $\lambda = 0$



(b) stochastic gradient ascent with subset size 65 and $\lambda = 0$



(c) regular gradient ascent with $\lambda = 20$



(d) stochastic gradient ascent with subset size 65 and $\lambda = 20$

Figure 4: Error vs Iteration for Regularized Update Rule for a Training Data Set with Size $M = 585$.

Step 5: To compare the results with the decision tree algorithm, the performance metrics are averaged over different folds. Table 1 shows the performance of the decision tree implemented in Machine Problem I and Table 2 demonstrates the performance of the Logistic Regression classifier with $\lambda = 20$ and regular gradient ascent.

Table 1: Average Performance Metrics for Decision Tree

Class Pair	**Accuracy mean/std**	**Precision mean/std**	**Recall mean/std**	**F_measure mean/std**
(4,15)	1/0	1/0	1/0	1/0
(5,14)	0.975/0.020	0.954/0.102	0.973/0.032	0.959/0.057
(7,12)	0.869/0.077	0.861/0.076	0.833/0.136	0.839/0.080

Table 2: Average Performance Metrics for Logistic Regression

Class Pair	**Accuracy mean/std**	**Precision mean/std**	**Recall mean/std**	**F_measure mean/std**
(4,15)	0.950/0.111	0.952/0.088	0.983/0.050	0.967/0.069
(5,14)	0.896/0.133	0.918/0.121	0.925/0.084	0.919/0.095
(7,12)	0.832/0.064	0.742/0.123	0.892/0.094	0.801/0.083

It is clear that the decision tree algorithm outperforms the logistic regressor. The performance of different classifiers depends on the data set and the distribution of the attributes of the data points. The results show that the decision tree classifier is able to distinguish the classes more accurately.