Technical Updates (Task-4)

Ved Surendra Thakur (B19ME090)

January 31, 2021

1 Hyper-parameter tuning in Logistic Regression

Sometimes one can see useful difference in performance or convergence with different solvers.

solver in ['newton-cg','lbfgs', 'liblinear,'sag','saga']

Regularization (penalty) can sometimes be helpful

penalty in ['none','L1','L2','elasticnet']

The C parameter controls the penalty strength which can also be effective.

C in [100, 10, 1.0, 0.01]

Parameters:

Penalty: The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization. The 'liblinear' solver supports both L1 and L2 regularization. The Elastic-Net regularization is only supported by the 'saga' solver.

dual: Dual formulation is implemented for l2 penalty with liblinear solver. We should prefer dual=False when n(samples) ¿ n(features)

tol: It is tolerance for stopping criteria.

C: This is one of the most important hyper-parameters in logistic regression. It is inverse of regularization strength therefore small the value better the regularization.

fit intercept: specifies if a constant (a.k.a. bias or intercept) that should be added to the decision function.

intercept scaling: It is useful only when 'liblinear' solver is used.

class weight: Weights associated with classes in the form class label: weight. If it is not given, then all classes are supposed to have equal weight.

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n(samples) / (n(classes) * np.bincount(y)).

These weights will be multiplied with sample weight, if sample weight is specified.

random state: we can use to shuffle the data when solver is sag, saga or liblinear.

max iter: maximum no. of iterations fir solvers to converge.

warm start: when it is set to true, it just reuses the solution of the previous call to fit as initialization, otherwise it just erases the previous solution.

n jobs: Number of CPU cores used when parallelizing over classes if multi class='ovr'". This parameter is ignored when solver is liblinear.

multi class: if 'over' option is chosen then, a binary problem is fit for each label. For 'multinomial' the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary.

Parameters which I will use for my model:

I will use Grid Search to find out which hyperparamters that perform best for logistic regression. But regarding the solver, I would decide it manually:

If my data set is small, I will use lib-linear and if my data se is large; I will use sag and saga as these are for larger datasets.

If my dataset has multiple classes I will use 'newton-cg', 'sag', 'saga' and 'lbfgs'as these handle multinomial loss. liblinear is limited to one verses rest classes type.

newton-cg', 'lbfgs', 'sag' and 'saga' handle L2 or no penalty whereas liblinear' and 'saga' also handle L1 penalty

Also 'sag' and 'saga' fast convergence is only guaranteed on features with approximately the same scale therefore we need to perform normalization in this case.

2 Under-fitting and Over-fitting

Over-fitting is an important problem in any classifier problems and this needs to be addressed, as the accuracy would drop drastically when testing on any new data. To avoid over-fitting a regression model, one should draw a random sample that is large enough to handle all the terms that one expects to include in the model. If the area under ROC is more than 95 percent then it's most likely that the model has over-fitted. If we are using too many predictors then such biases might occur. Therefore we need to limit the number of predictors we are using. After applying GMM we will use only one parameter predictor, either EM or VB-EM not both. Also an over-fitted logistic regression has large variance, means decision boundary changes largely for small change in variable magnitude.

for e.g.

$$h(X) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2) \tag{1}$$

$$h(X) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2) \tag{2}$$

$$h(X) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3) \tag{3}$$

Here Eqn(1) leads to problems of under-fitting whereas Eqn(3) leads to problems of over-fitting.An over fitted regression model have too many features while under-fit model has very less no. of features. Therefore I will use model similar to Eqn(2) which has balance between the variance and bias.

Another ways to solve the problems of over-fitting are:

1) penalized maximum likelihood estimation (ridge regression, elastic net, lasso, etc.)

2) we will use informative priors with a Bayesian model.

3 Gradient Ascent

Learning rate for the Gradient Ascent needs to be changed. Since the function is not concave and we have a single maxima.

Since having faster learning rate would affect the training error and using a small learning rate would lead to stucking with a high training error. Therefore we should have a medium learning rate. This will ensure that we reach the required maxima and not miss it since we don't have multiple maximas.

We can also include exponentially weighted average of the prior updates to the weight when the weights are updated. This change to stochastic gradient descent is called "momentum" and adds inertia to the update procedure.

According to me, we should also perform batch gradient ascent. The cost is calculated for an algorithm over the entire training data set for each iteration of the gradient descent algorithm. One iteration of the algorithm is called one batch and this form of gradient descent is referred to as batch gradient descent.

4 Alternative Method

Another method which I was thinking is, if the number of missing features is less than 10 percent of the total data then we can drop those data points which have; one or two missing data points. This might increase the accuracy of the problem and we would have less computations too. Also this complex math involving GMM and EMs would not be needed.

Some people might argue that we cannot do this; since we divide the complete data into training and testing samples we can have data points with one or two missing features in the testing sample. In such a case they would sort all data points having missing features as testing data and data points having complete feature vector as training data.