Logistic Regression with concepts of Robustness and Stability

Peijun Xu
Massachusetts Institute of Technology
Operations Research Center
MA, USA
xup@mit.edu

Denis Sai
Massachusetts Institute of Technology
Operations Research Center
MA, USA
dsai@mit.edu

April 13, 2021

Abstract.

Regularization is used in linear regression models as a technique to prevent overfitting by adding a penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values. However, as recently demonstrated, the real reason for adding a regularization term is the idea of achieving robustness, which means the ability to tolerate perturbations in data sets. We found that the implementation of Logistic Regression in the state-of-art machine learning package, Scikit-learn, simply adds a regularization term at the end of its loss function. We believe such implementation is suboptimal as it doesn't account for the true uncertainty inside data sets and we'd like to fix this implementation so that it achieves robustness. We show that out model show more robust model coefficients on all presented data sets. Moreover, we want to further improve the performance by incorporating the idea of stability. The paper would demonstrate our implementations of 3 different models: Robust Logistic Regression, Stable Robust Logistic Regression as well as Stratified Stable Robust Logistic Regression. It would also show the comparisons between the results of the 4 models (3 models we implemented plus Conventional Logistic Regression in scikit-learn) fitting on 4 real-life data sets with different sample sizes and feature sizes. Our implemented robust models we show average improvement of 6.5% in out-of-sample ROC AUC across 4 data sets comparing with conventional Scikit-learn approach.

1 Introduction

One of the most common ways to solve a classification problem is to build a logistic regression. The linear classification model is defined as following:

$$a(x) = sign(< w, x > +w_0) = sign(\sum_{j=1}^{d} w_j x_j + w_0) \tag{1}$$

Where $X = R^{l*d}$ - feature set, $Y = \{-1, +1\}$ - binary response space, w - weights vector, w_0 - bias.

The error function corresponding to the linear classification model has the form:

$$L(a, X) = \frac{1}{l} \sum_{i=1}^{l} [y_i < w, x_i >< 0] \rightarrow min_w \tag{2}$$

The logistic regression formulation proceeds from an estimate of a linear classification model through an upper bound on the error functional so that the upper bound is a smooth function:

$$L(a, X) \leq \overline{L}(a, X) \tag{3}$$

As a smooth upper bound for the logistic regression problem, the logistic loss function is used. In addition, a regularization term is often added to the formulation in the same way as it is added to a linear regression problem. We will write down the final formulation of the problem, which is used in this form, including in one of the most famous packages for machine learning, Sklearn:

$$min_w \sum_{i=1}^{l} log(1 + e^{-y_i w^T x_i}) + \frac{1}{2\sigma} w^T w \tag{4}$$

Note that such a regularization addition is caused by a simple correspondence with the linear regularization model. However, the real reason for adding regularization to linear regression is that it achieves robustness(Bertsimas and Copenhaver, 2018). Similar to linear regression, adding regularization to logistic regression can help achieve robustness if the regularization is added in the right place(Bertsimas and King, 2017). The robustness in features in logistic regression is:

$$max_{\beta,\beta_0} min_{\delta X \in U_x} - \sum_{i=1}^{l} (1 + e^{-y_i \beta^T (x_i + \delta x_i) + \beta_0}) \tag{5}$$

Robust counterpart to 10 is:

$$max_{\beta,\beta_0} - \sum_{i=1}^{n} log(1 + e^{-y_i (\beta^T x_i + \beta_0) + p\beta^2}) \tag{6}$$

Where L_{q^*} is the dual norm of L_q.

Thus, we get two different logistic regression formulations, but only the latter guarantees robustness in features. However, the most popular machine learning package, Sklearn, still uses the first formulation. This implies one of the objectives of our article - we want to compare the out of sample results of the two formulations on real data using metrics like ROC AUC, Accuracy, TPR and FPR. If we see a significant increase in metrics in the latest robust approach, that would be a good reason to redefine the problem in the most popular machine learning library today.

Note that the standard procedure for evaluating out of sample metrics consists of randomly splitting the data into train, validation, and test, where, first, at the validation stage, the hyperparameters (in our case, the regularization coefficient) are tuned, after which the model is retrained on the combined train and validation data with selected hyperparameters. With the help of the final model, an assessment is made on a deferred sample, a test set, which is the final benchmark of the effectiveness of the constructed model. Note that since the data splitting occurs in a random way, depending on the case, we can get unstable out of sample results and the model parameters, together with regularization, can differ significantly

depending on the random data splitting. To solve this problem, a method for creating a stable regression was proposed (Bertsimas and Dunn, 2019), the methodology of which we also want to use in our analysis. If we can see that with added stability we get better results on lazy sampling, we can also suggest adding this to the most popular machine learning library.

Since in the current literature stable regression is presented only for the linear case, below we will describe in detail the process of constructing a stable logistic regression together with the model tuning methodology.

2 Models

In general, Logistic Regression can be written as a convex, non-linear optimization problem with an objective of maximizing its logistic likelihood:

$$max_{\beta,\beta_0} \prod_{i=1}^{n} P(Y_i = y_i) = max_{\beta,\beta_0} - \sum_{i=1}^{n} log(1 + e^{-y_i(\beta^T x_i + \beta_0)}) \tag{7}$$

In our analysis, we will use 4 models that aim to improve the performance of Logistic Regression: Regularized Logistic Regression in Sklearn, Robust Logistic Regression, Stable Robust Logistic Regression and Stratified Stable Robust Logistic Regression. We will describe each of the models below.

2.1 Conventional Logistic Regression in Sklearn (CLR)

The Logistic Regression algorithm in the state-of-art machine learning package Sklearn implemented the l2 regularization by simply adding the regularized term at the end of the objective function as the following:

$$min_w LogLoss(w) = min_w \sum_{i=1}^{l} log(1 + e^{-y_i w^T x_i}) + \frac{1}{2\sigma} w^T w \tag{8}$$

Although this implementation of regularization does slightly improve the out-of-sample performance of logistic regression by introducing some bias into the model to reduce the variance, we find it hard to interpret the term in a more sensible way.

2.2 Robust Logistic Regression (RLR)

Similar to linear regression, the real reason for adding regularization to logistic regression is to achieve robustness to account for the uncertainty contained in the data. Therefore, for the uncertainty set in the features:

$$u_x = \left\{ \Delta X \in R^{n \times p} \mid \|\Delta X_i\|_q \leq \rho,\ i = 1, ..., n \right\} \tag{9}$$

Robustifying 7 against the above set yields:

$$max_{\beta,\beta_0} min_{\Delta X \in U_x} - \sum_{i=1}^{n} (1 + e^{-y_i \beta^T (x_i + \Delta x_i) + \beta_0}) \tag{10}$$

From that perspective, the right place to add regularization term so that it incorporates robustness is the following:

$$LogLoss = \sum_{i=1}^{n} log(1 + e^{-y_i(w^T x_i) + \lambda w^T w}) \tag{11}$$

We can formulate Robust Logistic Regression as a convex optimization problem:

$$max_{\beta,\beta_0} - \sum_{i=1}^{n} log(1 + e^{-y_i(\beta^T x_i + \beta_0) + p\beta^2}) \tag{12}$$

and apply Cutting Plane algorithm to solve it.

$$f(\beta, \beta_0) = -\sum_{i=1}^{n} log(1 + e^{-y_i(\beta^T x_i + \beta_0) + \rho\beta^T\beta}) \tag{13}$$

$$\Delta f(\beta, \beta_0) = -\sum_{i=1}^{n} \frac{(2\rho\beta - y_i x_i)e^{-y_i x_i(\beta^T x_i + \beta_0) + \rho\beta^T\beta}}{1 + e^{-y_i x_i(\beta^T x_i + \beta_0) + p\beta^2}} \tag{14}$$

Algorithm 1: `Cutting Plane algorithm for Robust Logistic Regression`

Input: Data X,
ρ regularization parameter
ϵ convergence parameter
b_0 initialized weights (with our without warm-start)
t = 0
μ = 0
Result: $\beta_{optimal}, \beta_{0optimal}$

while $f(\beta_t) - \mu_t > \epsilon$ **do**
  solve: $\min_\beta \mu$
  s.t. $\mu \geq 0$
  $\mu \geq f(\beta_i) + \nabla f(\beta_i)^T(\beta - \beta_i)\ \forall i = 1...t$
  $t \leftarrow t + 1$;
end
return β_{t+1}, μ_{t+1} ;

2.3 Stable Robust Logistic Regression (SRLR)

Continuing from Robust Logistic regression, we need to add stability condition in the form of adding binary variable z_i for each observation indicating whether we add this observation to our train set or not. We can do that in the following way:

$$\begin{array}{lll} \min\limits_{\beta,\beta_0} & \frac{1}{k} max_z \sum_{i=1}^{l} z_i log(1 + e^{-y_i(\beta^T x_i + \beta_0) + p\beta^2}) & \\ \text{s.t.} & \sum_{i=1}^{l} z_i = k & \\ & z_i \in \{0, 1\}\ \forall i \in \{1, ..., l\} & \end{array} \tag{15}$$

As far as we can see, this formulation does not fit the description of the Mixed Integer Optimization problem (MIOP), so we cannot directly use, for example, the simplex method to solve this problem. If we turn to the original idea of stable regression, the authors used duality to transform the nonlinear loss function to linear constraints, which made it possible to use standard MIOP solution methods. In our case, we notice that the dual problem will have nonlinear constraints, which will also not fit the MIOP and further complicates our task.

Another way of solving the problem is the previously described approach in (Bertsimas et al., 2020) using Cutting Plane for problems with a convex loss function. First, let's prove that our loss function in the 24 formulation is convex.

To do this, let's first note that the logistic regression cost function is convex that was proved in previous research. It follows from this that all internal logarithms in our loss function are convex. Then, inside the maximization problem, we have a sum of convex functions. Let's prove that sum of convex functions is a convex function:

We know that for convex functions it is true that:

$$f_1(a * x + (1 - a) * y) \leq a f_1(x) + (1 - a) f_1(y)$$

$$f_2(a * x + (1 - a) * y) \leq a f_2(x) + (1 - a) f_2(y)$$

Then:

$$f_1(a * x + (1 - a) * y) + f_2(a * x + (1 - a) * y) \leq a f_1(x) + (1 - a) f_1(y) + a f_2(x) + (1 - a) f_2(y)$$

If we rewrite it with $f_3 = f_1 + f_2$, we have:

$$f_3(a * x + (1 - a) * y) \leq af_3(x) + (1 - a)f_3(y)$$

Because Jensen's inequality is satisfied, the function is convex. The same will be true for the sum of a finite number of convex functions.

So now we know that inside maximization we have a convex function. In our optimization problem, we need to find the maximum among several convex functions. Therefore, in order to use the proposed Cutting Plane method, now we need to prove that the maximum of several convex functions is also a convex function.

Lets identify each set of convex functions as $\{f_1(x), f_2(x), ..., f_n(x)\}$ and maximization over set as $f(x) = max(f_1(x), f_2(x), ..., f_n(x))$.

Again, for each individual convex function Jensen's inequality holds:

$$f_i(a * x + (1 - a) * y) \leq af_1(x) + (1 - a)f_i(y) \tag{16}$$

Let's take maximum of both sides, we have:

$$max_i(f_i(a * x + (1 - a) * y)) \leq max_i(af_i(x) + (1 - a)f_i(y)) \tag{17}$$

Notice that:

$$max_i(af_i(x) + (1 - a)f_i(y)) \leq max_i(af_i(x)) + max_i((1 - a)f_i(y)) \tag{18}$$

$$0 \leq max_i(af_i(x)) + max_i((1 - a)f_i(y)) - max_i(af_i(x) + (1 - a)f_i(y)) \tag{19}$$

Let's add 17 to 19:

$$max_i(f_i(a * x + (1 - a) * y)) \leq max_i(af_i(x)) + max_i((1 - a)f_i(y)) \tag{20}$$

Because we know that $f(x) = max(f_1(x), f_2(x), ..., f_n(x))$, we can rewrite it as:

$$f_(a * x + (1 - a) * y) \leq af(x) + (1 - a)f(y) \tag{21}$$

And this is exactly Jensen's inequality, so we proved. Now we can apply the Cutting Plane algorithm described below.

Let's state that:

$$f(\beta, \beta_0) = \frac{1}{k} max_z \sum_{i=1}^{l} z_i log(1 + e^{-y_i(\beta^T x_i + \beta_0) + p\beta^2}) \tag{22}$$

$$\nabla f(\beta, \beta_0) = \frac{1}{k} \frac{z_i(2p\beta - y_i x_i)e^{-y_i x_i(\beta^T x_i + \beta_0) + p\beta^2}}{e^{-y_i x_i(\beta^T x_i + \beta_0) + p\beta^2} + 1} \tag{23}$$

As described in (Bonami et al., 2008), we can find solution to our primal by iteratively constructing a piece-wise linear lower approximations.

Algorithm 2: `Cutting Plane algorithm for Stable Robust Logistic Regression`

Input: Data X,
p regularization parameter
ϵ convergence parameter
b_0 initialized weights (with our without warm-start)
t = 0
μ = 0
Result: $\beta_{optimal}, \beta_{0optimal}$

while $f(w_t) - \mu_t > \epsilon$ **do**
  solve: $\min_w \mu$
  s.t. $\mu \geq 0$
  $\mu \geq f(w_i) + \nabla f(w_i)^T (w - w_i)\ \forall i = 1...t$
  $t \leftarrow t + 1$;
end
return w_{t+1}, μ_{t+1} ;

2.4 Stratified Stable Robust Logistic Regression (SSRLR)

Based on the problems that we will see in the part of the results, we added a stratification model, with an additional constraint on the percentage of classes in the partition. The task is formulated as follows:

$$\begin{array}{lrcl} \min\limits_{\beta,\beta_0} & \frac{1}{k} max_z \sum_{i=1}^{l} z_i log(1 + e^{-y_i(\beta^T x_i + \beta_0) + p\beta^2}) & & \\ \text{s.t.} & \sum_{i=1}^{l} z_i & = & k \\ & z_i & \in & \{0,1\}\ \forall i \in \{1,...,l\} \\ & \frac{1}{k}\sum_{i=1}^{l} \frac{y_i+1}{2} z_i & = & \frac{1}{n-k}\sum_{i=1}^{l} \frac{y_i+1}{2}(1 - z_i) \end{array} \tag{24}$$

The only difference from SRLR is the added constraint to the same percentage of class 1 in our breakdown. Thus, now we are trying to find the most complex train set with the condition that the samples must be stratified (this is due to the last added condition).

The task also remains convex, which allows us to use the Cutting Plane algorithm again. The approach to finding a solution remains exactly the same as in 2.

3 Experimental Framework

To evaluate the performances of the models, we fitted 4 models on 4 datasets from the UCI Machine Learning Repository. The 4 datasets we chose have sample sizes ranging from n=80 to n=3658 and feature sizes ranging from 10 to 38 so that we can see the performance change along with different sample and feature sizes

3.1 Data

In the preprocessing step, we deleted the rows with missing data, normalized all numerical features using z-score $z = \frac{x - \bar{x}}{s}$ and one-hot coded all categorical features. After preprocessing, we have:

- Caesarian Section Classification Dataset Data Set ($n = 80$, $p = 10$)
- Monk Data Set ($n = 432$, $p = 12$)
- Credit Approval Data Set ($n = 690$, $p = 38$)
- Framingham Data Set ($n = 3658$, $p = 18$)

3.2 Methodology

For all our models, we split the data 80/20 in train/test using the same seed, then we used the same test sets for all our models. We divided the analysis into two parts:

- Comparative analysis of key performance metrics (ROC AUC, accuracy) with fixed split into train and validation:

1. For CLR and RLR, we validated the results with a random split of the data into 80/20 in train/-validation, after which we chose the best parameter for Ridge regularization by ROC AUC on the validation set
2. For the Stable implementations (SRLR, SSRLR), we first chose the best complex train set (and the lightest validation set) along with the best hyperparameter. We also used an 80/20 ratio for this task.

- Identification of stability of model coefficients:
  1. For each dataset, we divided the data in proportions 50/50, 60/40, 70/30, 80/20, 90/10 into a train / validation split for all our models, after which we validated the models in each of the splits.
  2. For optimization approach RLR, we learned the optimal coefficients β from these training/validation splits, and then β that yielded the highest ROC AUC on the validation set was selected. These coefficients are applied to the held-out test set.
  3. After that, for each dataset and for each of the models, we estimated the standard deviation of each of the parameters for 5 splits, after which we compared the RLR and CLR for each of the standard deviations of the regressors.

This way we will be able to evaluate whether our optimization methods have become better performing and whether our models have really become more robust.

4 Results

First, we would like to highlight the value of using Stratified Stable Robust Logistic Regression. In our experiments, we saw that if the samples are imbalanced when the percentage of data assigned to the validation set we are choosing is lower than the percentage of the majority class in the dataset. For example, if our stable constraint is on 50% level, but 80% pf our data predicts -1, and all samples are chosen so that they predict -1. In this case, the model really chooses the simplest validation set (since there are only representatives of one class), but such a partition does not help us to identify the best model, since in order to estimate ROC AUC (our main metric), both samples must contain representatives of both classes.

Moreover, depending on the regularization parameter, sometimes with the same percentage of data in the validation set, we observed the appearance of examples of both classes in both samples, which made it possible to continue the analysis. Thus, in some cases, we could build models and validate them depending on the hyperparameters, but for each of the datasets, some sets of hyperparameters made it impossible to continue the analysis, which reduced the space for choosing hyperparameters at the validation stage to a reduced subset (which, it should be noted, is always was non-empty; that is, for each percentage split into train and validation, we could find at least one value for the regularization parameter at which the stable model separated classes between samples).

As a solution to this problem (a shorthand subset of hyperparameter selection). we propose a new method - SSRLR. Its only difference from SRLR is the addition of stratification by stipulating that the samples we choose have to have a distribution of labels that is similar to the larger dataset. Next, we'll look at the results of our models.

We have implemented all the outlined models in Julia, except for the CLR, which was implemented in Python. For each of the model implementations, we reported the ROC AUC, Accuracy, True positive rate (TPR) and False Positive Rate (FPR) indicators, the results are presented in the Table 1.

Table 1: Key Performace Metrics for CLR, RLR, SRLR, SSRLR

Table 2: Percentage of coefficients B where model showed less variability

Dataframe	CLR	RLR
Caesarian	11.1%	**88.9%**
Monks	18.2%	**81.8%**
Credit Screening	16.2%	**83.8%**
Framingham	35.3%	**64.7%**

Std of params are computed as $std(w_i^{50/50}, w_i^{60/40} w_i^{70/30} w_i^{80/20} w_i^{90/10})$ for each i and then each std for each coefficient comparec between CLR and RLR.

Data Set	Model	n	p	ROC AUC	Accuracy	TPR	FPR
Caesarian	CLR	80	10	0.65	0.625	**0.8**	0.67
Caesarian	RLR	80	10	0.65	0.625	**0.8**	0.67
Caesarian	SRLR	80	10	0.67	**0.75**	**0.8**	0.33
Caesarian	SSRLR	80	10	**0.70**	**0.75**	0.7	**0.17**
Monks	CLR	432	12	0.67	0.76	0.52	**0**
Monks	RLR	432	12	0.68	0.75	0.52	0.02
Monks	SRLR	432	12	0.73	0.76	0.61	0.16
Monks	SSRLR	432	12	**0.75**	**0.78**	**0.71**	0.14
Credit Screening	CLR	690	38	0.956	0.902	0.933	0.125
Credit Screening	RLR	690	38	0.952	0.902	0.917	0.11
Credit Screening	SRLR	690	38	0.958	**0.924**	0.961	0.125
Credit Screening	SSRLR	690	38	**0.962**	0.916	**0.983**	**0.138**
Framingham	CLR	3658	18	0.92	0.850	**0.977**	0.241
Framingham	RLR	3658	18	0.92	0.821	0.719	**0.105**
Framingham	SRLR	3658	18	0.92	0.857	0.930	0.190
Framingham	SSRLR	3658	18	0.92	**0.859**	0.915	0.180

As we can see, the optimized models performed better than the Conventional Logistic Regression implemented in Sklearn. Improvement in ROC AUC ranges from 0% to +10% compared to CLR. The best results (at least not inferior to CLR results) everywhere show SSRLR. This indicates that the addition of the stability property really allows you to split the sample so that the model parameters will better describe the test set. Note that SRLR metrics are almost always inferior to SSRLR metrics. This is due to the fact that for SRLR, we could not use the entire hyperparameter space, which cut off some potentially effective hyperparameter combinations.

Note that in other key metrics, our optimization methods also outperformed the CLR performance. SSRLR showed an increase in Accuracy from +1% to +20%. Note that the optimization model showed a greater gain on a small dataset (n = 80), while it showed less gain on a large dataset (n 4000). All this indicates that the use of stable and robust models makes sense.

In addition, since one of our tasks is not only to prove the improved performance but also to improve the robustness of the RLR in comparison with the CLR, we analyzed the robustness of the parameters depending on the percentage of data in the validation set; the results can be seen in the table 2, which leads from tables 3 4 5 6.

As far as we can see, for all the presented datasets and for most of the regressors, we have reduced the standard deviation of the model weights, which indicates the stabilization of the model coefficients. Note that our robust model shows the greatest effect of reducing the standard deviation on small datasets (more than 80% of reduced standard deviations), when on a large dataset the percentage of reduction is lower, 65%, which still remains within the framework of the reduction for most regressors.

Thus, we get a more robust model that is easy to implement in Sklearn. Even our simplest RLR model shows an improvement over the current CLR implementation in Sklearn both in terms of key metric performance and robustness of the coefficients.

5 Conclusion

In conclusion, we've provided a theoretical framework for solving Robust Logistic Regression, Stable Logistic Regression and Stratified Stable Robust Logistic Regression. Specifically, we proved convexity for the objective functions which is a prerequisite before applying Cutting Plane method and described algorithms in detail as well as the modification we made to improve the algorithms' computation efficiency. We have also implemented these algorithms and applied them on 4 real-life datasets with different sizes and

compared their results along with the results of Conventional Logistic Regression in Scikit-learn. Based on these results, we can see that correct implementation of robustness and incorporation of stability provide an edge over the state-of-art Conventional Logistic Regression with more than 6% increase in out-of-sample prediction performances and decrease in variability of more than 65% of estimated coefficients and we found that Stratified Stable Robust Logistic Regression achieves the best overall performance. However, one major drawback of our implementations is that the vanilla Cutting Plane method doesn't scale very well and takes hours to solve the dataset with n=30000, p=32 due to the long time spent on computing gradients. Therefore, for future work, we'd like to improve the computational time of our algorithms using methods like Stochastic Cutting Plane so that we can explore datasets with larger sizes and further solidify our findings as well as add more experiments on real world datasets so that we will be able to prove efficiency on more balanced and unbalanced data sets with different sizes.

References

Bertsimas, D. and M. S. Copenhaver (2018). Characterization of the equivalence of robustification and regularization in linear and matrix regression. *European Journal of Operational Research 270*(3), 931 – 942.

Bertsimas, D. and J. Dunn (2019). Machine learning under a modern optimization lens. pp. 333–349.

Bertsimas, D. and A. King (2017, 08). Logistic regression: From art to science. *Statist. Sci. 32*(3), 367–384.

Bertsimas, D., J. Pauphilet, and B. V. Parys (2020). Sparse classification: a scalable discrete optimization perspective.

Bonami, P., L. Biegler, A. Conn, G. Cornuéjols, I. Grossmann, C. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter (2008, May). An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization 5*(2), 186–204.

6 Appendix

Table 3: Comparison of weights' standard deviation across splits for Caesarian Data Set

Dataframe	Regressor	Ridge LG	Robust LG
Caesarian	w1	0.079	**0.03**
Caesarian	w2	0.188	**0.078**
Caesarian	w3	0.259	**0.049**
Caesarian	w4	0.224	**0.044**
Caesarian	w5	0.328	**0.056**
Caesarian	w6	0.289	**0.059**
Caesarian	w7	0.158	**0.027**
Caesarian	w8	0.179	**0.034**
Caesarian	w9	**0.128**	0.693

Table 4: Comparison of weights' standard deviation across splits for Monks Data Set

Dataframe	Regressor	Ridge LG	Robust LG
Monks	w1	0.144	**0.034**
Monks	w2	0.127	**0.078**
Monks	w3	0.065	**0.04**
Monks	w4	0.134	**0.028**
Monks	w5	**0.053**	0.061
Monks	w6	0.104	**0.056**
Monks	w7	0.185	**0.031**
Monks	w8	0.232	**0.02**
Monks	w9	**0.097**	0.414
Monks	w10	0.088	**0.06**
Monks	w11	0.152	**0.138**

Table 5: Comparison of weights' standard deviation across splits for Credit Screening Data Set

Dataframe	Regressor	Ridge LG	Robust LG
Credit Screening	w1	0.067	**0.016**
Credit Screening	w2	0.1	**0.026**
Credit Screening	w3	0.092	**0.03**
Credit Screening	w4	**0.076**	0.121
Credit Screening	w5	0.084	**0.067**
Credit Screening	w6	0.263	**0.217**
Credit Screening	w7	0.097	**0.096**
Credit Screening	w8	0.139	**0.064**
Credit Screening	w9	0.113	**0.072**
Credit Screening	w10	0.139	**0.065**
Credit Screening	w11	0.113	**0.072**
Credit Screening	w12	0.199	**0.041**
Credit Screening	w13	0.111	**0.058**
Credit Screening	w14	0.178	**0.064**
Credit Screening	w15	0.046	**0.008**
Credit Screening	w16	**0.1**	0.127
Credit Screening	w17	0.371	**0.07**
Credit Screening	w18	0.162	**0.062**
Credit Screening	w19	0.175	**0.01**
Credit Screening	w20	**0.142**	0.153
Credit Screening	w21	0.213	**0.077**
Credit Screening	w22	0.164	**0.034**
Credit Screening	w23	0.18	**0.057**
Credit Screening	w24	0.123	**0.117**
Credit Screening	w25	0.259	**0.049**
Credit Screening	w26	0.225	**0.06**
Credit Screening	w27	0.23	**0.132**
Credit Screening	w28	0.12	**0.085**
Credit Screening	w29	0.084	**0.031**
Credit Screening	w30	**0.104**	0.12
Credit Screening	w31	**0**	0.001
Credit Screening	w32	0.101	**0.029**
Credit Screening	w33	**0.056**	0.236
Credit Screening	w34	**0.12**	0.149
Credit Screening	w35	0.14	**0.028**
Credit Screening	w36	0.064	**0.051**
Credit Screening	w37	0.064	**0.057**

Table 6: Comparison of weights' standard deviation across splits for Framingham Data Set

Dataframe	Regressor	Ridge LG	Robust LG
Framingham	w1	**0.029**	0.03
Framingham	w2	**0.06**	0.075
Framingham	w3	0.073	**0.043**
Framingham	w4	0.301	**0.029**
Framingham	w5	**0.069**	0.075
Framingham	w6	0.135	**0.033**
Framingham	w7	0.081	**0.019**
Framingham	w8	0.032	**0.022**
Framingham	w9	**0.014**	0.028
Framingham	w10	0.026	**0.015**
Framingham	w11	0.033	**0.032**
Framingham	w12	0.047	**0.021**
Framingham	w13	0.026	**0.015**
Framingham	w14	0.047	**0.013**
Framingham	w15	0.038	**0.022**
Framingham	w16	**0.125**	0.288
Framingham	w17	**0.13**	0.317