Comparing Standard Logistic Regression and Logistic Regression with Coordinate Descent Optimization

Mikołaj Mróz, Krzysztof Sawicki, Tymoteusz Urban

March 2025

Contents

1 Introduction

Logistic regression is one of the most widely used models in binary classification due to its simplicity and interpretability. However, the choice of optimization techniques and the inclusion of regularization can significantly impact its performance, especially when dealing with high-dimensional data. In this study, we compare the performance of three logistic regression variants:

1. **Standard logistic regression without regularization**,
2. **Logistic regression with L_1 regularization (LASSO)**,
3. **Logistic regression with Coordinate Descent (CCD)**, a custom implementation designed to optimize for both L_1 and L_2 penalties.

The evaluation was conducted on a set of **four real-world datasets** and **artificially generated datasets**, with performance assessed using standard classification metrics such as ROC AUC, Recall-Precision AUC, F-measure and Balanced Accuracy. The goal of this comparison was to understand the trade-offs between these approaches, particularly in terms of their predictive accuracy, sparsity of the resulting models, and computational efficiency. This paper provides insights into how regularization and optimization techniques can influence model performance across different types of datasets, offering guidance for selecting the right variant in real-world applications. All the used codes can be found in the repository on GitHub Repository.

2 Datasets

2.1 Parkinson's Disease Classification

The Parkinson's Disease Classification dataset, hosted on the UCI Machine Learning Repository, is designed to support research in detecting and classifying Parkinson's disease.The dataset consists of 754 features for 756 instances. The primary goal of this dataset is to classify individuals as either having Parkinson's disease or being healthy. The dataset was selected due to its high dimensionality relative to the number of observations. During preprocessing, highly correlated features were removed and standardization was applied.

2.2 SECOM

The SECOM dataset, another set hosted on the UCI Machine Learning Repository, contains data about semiconductor manufacturing and is used to analyze the impact of various sensor signals on production yield. Each row is a single production entity and the labels represent a simple pass/fail yield (0 for pass, 1 for fail) based on in-house testing. After preprocessing (removing missing values, dropping highly correlated columns, balancing target variable) we obtained dataset with 208 instances and 268 variables.

2.3 Arrhythmia Classification

The Arrhythmia dataset from the UCI Machine Learning Repository is designed to support research in the detection and classification of various cardiac arrhythmias. It originally comprises 452 instances with 279 features, making it a high-dimensional dataset where the feature count constitutes a significant proportion of the total observations. All variables are numerical, which simplifies preprocessing. During data preparation, highly correlated features were removed and standardization was applied to improve model performance.

2.4 Ionosphere Classification

The Ionosphere dataset is another resource from the UCI Machine Learning Repository, originally used to analyze radar returns for distinguishing between 'good' and 'bad' signals in the ionosphere. Since the original dataset did not meet the criterion that the number of variables should be at least 50% of the number of observations, we removed certain rows to enforce this condition. The final dataset comprises 68 instances with 34 features, thereby ensuring that the feature-to-observation ratio meets the required threshold. All variables are numerical and free from missing values. Preprocessing steps included standardization and, when necessary, the incorporation of dummy variables created from permuted copies of the original features.

2.5 Synthetic

During testing, we utilized synthetic datasets, which were generated using four parameters: p (class prior probability), n (number of observations), d (number of variables), and g (covariance decay factor). The generation procedure is as follows:

1. The target variable Y (0 or 1) is sampled from a Bernoulli distribution with class prior probability p.
2. Given $Y = 0$, the feature vector X follows a d-dimensional multivariate normal distribution with mean $(0, \ldots, 0)$ and covariance matrix S with $S[i, j] = g^{|i-j|}$, whereas for $Y = 1$, the feature vector X follows a d-dimensional multivariate normal distribution with mean $(1, 1/2, 1/3, \ldots, 1/d)$ and the same covariance matrix.
3. n samples are generated using the above procedure.

3 Model Implementation

The model implemented is a logistic regression with elastic net regularization, utilizing the coordinate descent (CCD) algorithm for optimization. It is based on implementation provided in the [Friedman et al., 2010]. Below is a description of the algorithm's implementation and applied optimizations.

3.1 Initialization

The `LogRegCCD` class is initialized with several key parameters:

- `lambda_min` and `lambda_max`: These parameters define the range of regularization strength values (λ) that will be evaluated. The lambda values are logarithmically spaced between these bounds.
- `num_lambdas`: The number of different lambda values to test during the optimization.
- `alpha`: This parameter controls the elastic net mixing ratio. An α value of 1 corresponds to LASSO (L1 regularization), while an α value of 0 corresponds to Ridge (L2 regularization). In the implementation, α is set to 0.2, a value between LASSO and Ridge.
- `coefficients`: The model's coefficients, including the intercept, are initialized to small random values drawn from a normal distribution.

3.2 Coordinate Descent Optimization

The core of the algorithm is the coordinate descent optimization, which is an iterative process. At each iteration, the coefficients are updated one by one based on the partial residuals. The following steps are taken for each feature during the optimization process:

- **Partial Residual Calculation**: For each feature j, the residuals are calculated by subtracting the predicted probabilities from the true labels, using the current set of model coefficients (except the j-th coefficient). This is done in the `fit()` function.
- **Gradient Calculation**: For each feature, the gradient of the log-likelihood with respect to that feature's coefficient is computed.
- **Regularization and Soft Thresholding**: Elastic net regularization is applied, where a combination of L1 and L2 penalties is used. The L1 penalty encourages sparsity, while the L2 penalty stabilizes the model. The soft-thresholding rule is applied to the gradient to update the coefficient.
- **Intercept Update**: After the coefficients are updated, the intercept is recalculated by averaging the difference between the true labels and the predicted probabilities.

3.3 Log-Loss Calculation

At each iteration, the log-loss is calculated. Log-loss measures the performance of the model by quantifying the difference between the predicted probabilities and the actual binary labels. The log-loss is calculated using the following formula:

$$\text{Log-Loss} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

where y_i are the true labels, and p_i are the predicted probabilities. Monitoring the log-loss helps track the convergence of the algorithm and the performance of the model during the training process.

3.4 Model Validation

Once the model is fitted, it is validated on a separate validation set using different performance metrics such as F1 score, precision, recall, and others. The `validate()` method calculates the chosen performance metric by comparing the predicted labels with the true labels.

3.5 Lambda Optimization

To find the optimal regularization parameter (λ), the `optimize_lambda()` method is used. This method performs a grid search over a range of lambda values, evaluating the model's performance on a validation set for each lambda. The lambda that results in the best performance (according to the selected metric) is chosen.

3.6 Visualization

The `plot()` method allows visualization of the performance metric (e.g., F1 score) as a function of lambda values, plotted on a logarithmic scale. Additionally, the `plot_coefficients()` method visualizes how the coefficients change with varying values of lambda.

3.7 Optimizations Applied

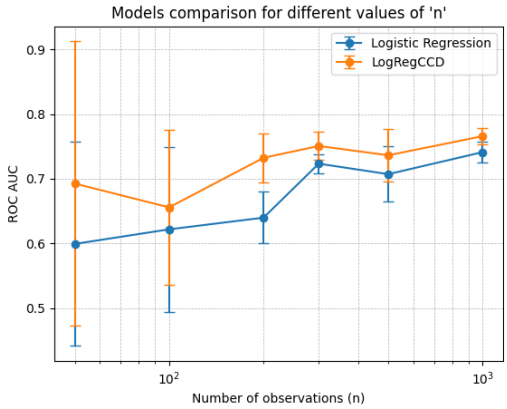
- **Coordinate Descent**: The use of coordinate descent allows for efficient optimization by updating one coefficient at a time, which is particularly suitable for models with a large number of features. This method is memory-efficient and allows the model to scale well with high-dimensional data. All operations are performed using NumPy operators, which vectorize the calculations and speed up the process.
- **Logarithmic Lambda Grid**: The lambda values are spaced logarithmically between `lambda_min` and `lambda_max`, ensuring that the model tests a wide range of regularization strengths. This allows for a more thorough exploration of the hyperparameter space.
- **Early Stopping Criteria**: While the implementation does not include explicit early stopping based on log-loss convergence, the log-loss output allows for monitoring the progress of the algorithm and making adjustments if necessary.

4 Benchmarks

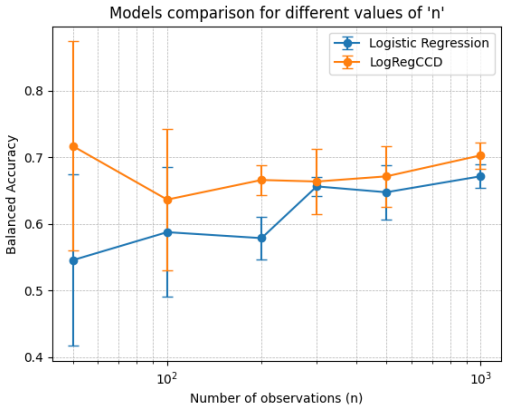
4.1 Synthetic dataset

We began testing our algorithm by conducting experiments on a synthetic dataset. We compared how the performance of CCD algorithm and logistic regression without regularization depends on the parameters n, p, g, and d. During testing single parameter, all other were set to default values ($n = 500$, $d = 50$, $p = 0.5$, $g = 0.5$). Training for each parameter variation was repeated 3 times, and the average Balanced Accuracy and ROC AUC scores were calculated.

Firstly, the impact of the number of observations on the performance of the algorithms was analyzed. In Figure 1, it can be seen that, in general, the performance of the algorithms improves as the number of observations increases. The only exception is for $n = 50$, but testing on just 17 instances (with the test set size being 30% of n) is not a sufficiently representative example, as such a small test size is highly vulnerable to variability.



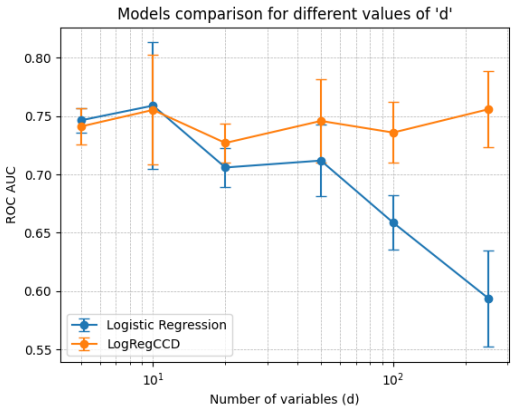
(a) ROC AUC



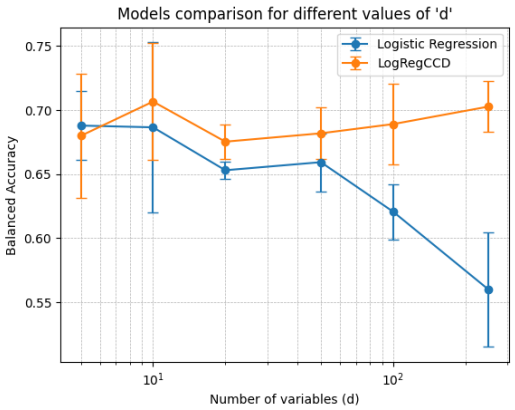
(b) Balanced Accuracy

Figure 1: Influence of parameter n

Secondly, we examined parameter d, that is the number of variables. The higher the number of variables, the greater the difference in performance between logistic regression and the CCD algorithm. Classic logistic regression, with no regularization, cannot properly handle that many variables, and its performance worsens as the number of variables increases (Figure 2).



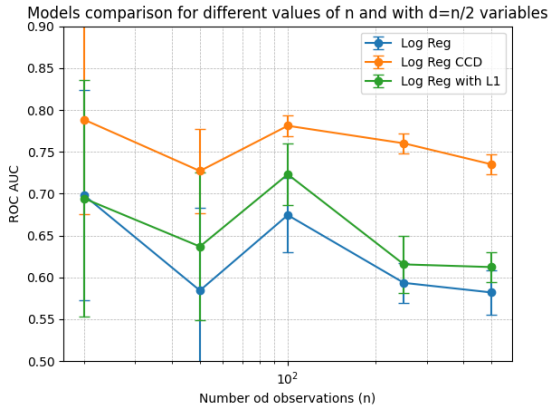
(a) ROC AUC



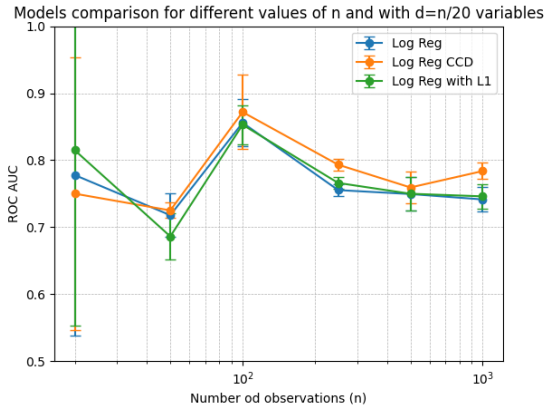
(b) Balanced Accuracy

Figure 2: Influence of parameter d

However, it is not about the absolute number of variables, bbut rather the ratio to the number of instances. We tested the algorithms with datasets generated for $d = n/2$ and $d = n/20$ (Figure 3). In the latter, there is basically no difference in performance between logistic regression and CCD algorithm. However, when number of variables equals 50% of number of instances, classic regression performs way worse and even adding L1 regularization does not help much.



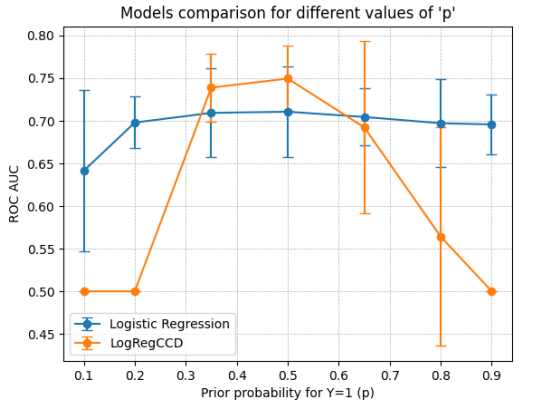
(a) Plot for $d = n/2$



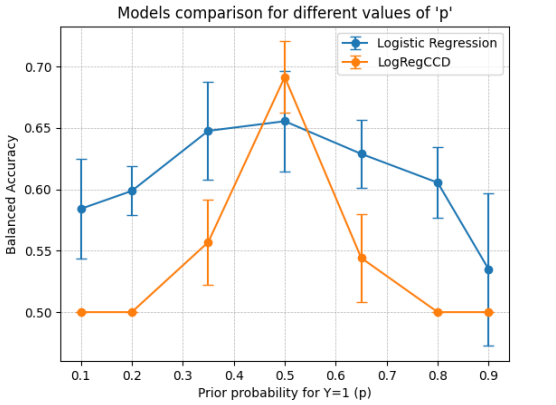
(b) Plot for $d = n/20$

Figure 3: Influence of parameter d

Another tested parameter was p - prior probability for $Y = 1$. Here the performance of CCD was bad for extreme values of p. For p being 0.1, 0.2, 0.8 and 0.9 the predictions were almost completely random. CCD algorithm works best, if the prior probability for $Y = 1$ is around 0.5. Results are shown in Figure 4.



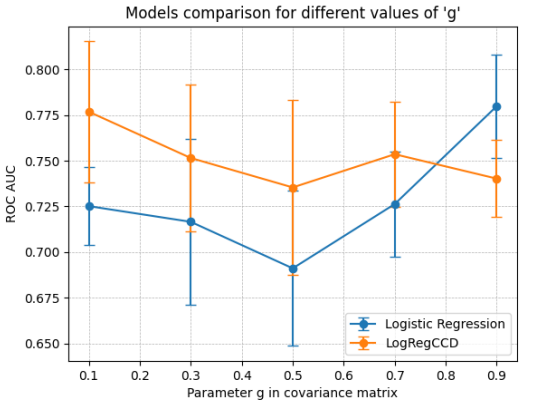
(a) ROC AUC



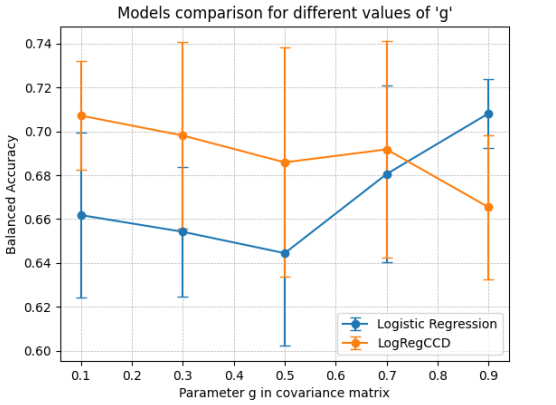
(b) Balanced Accuracy

Figure 4: Influence of parameter p

Finally, we tested parameter g, which could be interpreted as covariance decay factor. Conclusion from the Figure 5, is that when dealing with relatively highly correlated columns, CCD performance is really poor, while logistic regression handles it much better.



(a) ROC AUC



(b) Balanced Accuracy

Figure 5: Influence of parameter g

All experiments were conducted with λ optimized for the F1-score. However, scores for the CCD algorithm with $\lambda = 0$ were also evaluated. The average balanced accuracy score for all algorithm runs throughout the experiments on synthetic datasets was 0.644 (0.079), while for $\lambda = 0$, it was 0.617 (0.064). Furthermore, CCD with the optimized λ had a higher average balanced accuracy score than CCD with $\lambda = 0$ in 22 out of 24 cases.

4.2 Real datasets

ROC AUC is chosen as the primary metric because it is threshold-independent and robust in measuring a model's ability to discriminate between classes, making it ideal for comparing models across datasets with varying class distributions.

Table 1 shows the ROC AUC scores for:

- scikit-learn Logistic Regression (Unregularized)
- scikit-learn Logistic Regression with L1 Regularization (C=0.10)
- `LogRegCCD` (ROC AUC Optimization)

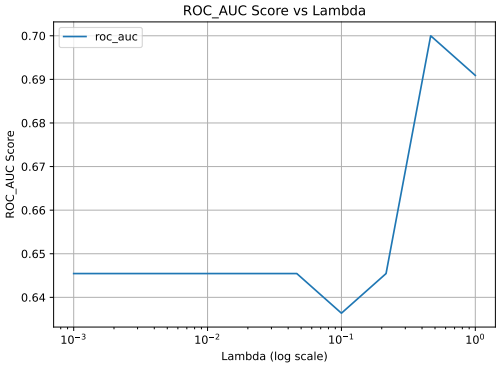
Dataset	Unregularized LR	L1-Regularized LR	`LogRegCCD`
Arrhythmia	0.6571	0.8734	0.8548
Ionosphere	0.6818	0.6000	0.7000
Secom	0.6058	0.7702	0.7571
Speech	0.8291	0.8523	0.8371

Table 1: ROC AUC scores for different models on real datasets.

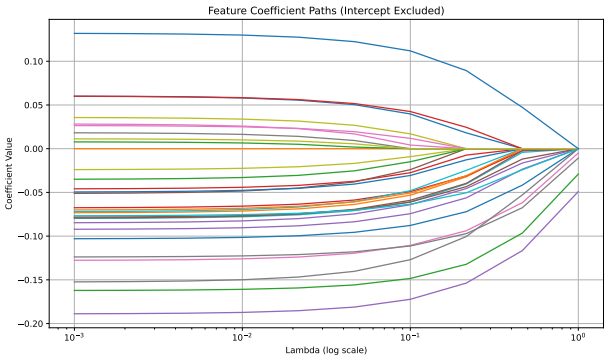
Best `LogRegCCD` lambda values: Arrhythmia: 0.100000, Ionosphere: 0.464159, Secom: 0.464159, Speech: 0.100000.

4.2.1 Ionosphere Detailed Analysis

Since our model outperformed the scikit-learn implementations on Ionosphere, we provide further analysis with two plots. Figure 6a shows ROC AUC vs. λ (log scale), indicating an optimal ROC AUC of 0.7000 at $\lambda = 0.464159$. Figure 6b displays coefficient paths as λ varies, illustrating how regularization induces sparsity and stabilizes feature weights.



(a) ROC AUC vs λ (log scale)



(b) Coefficient Paths

Figure 6: Ionosphere analysis with `LogRegCCD` (optimal $\lambda = 0.464159$).

Conclusions: For Arrhythmia and Secom, L1 regularization yields the highest ROC AUC, with `LogRegCCD` slightly lower. In Ionosphere, `LogRegCCD` outperforms the unregularized model, while L1 regularization is less effective. In Speech, all models perform similarly, with a slight edge for L1 regularization.

References

J. H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. doi: 10.18637/jss.v033.i01. URL https://www.jstatsoft.org/index.php/jss/article/view/v033i01.