Advanced Machine Learning
Project 1

Jakub Sawicki, Kinga Frańczak, Mateusz Wiktorzak

repository: https://github.com/kubek22/aml_project

1 Methodology

1.1 Selection and generation of datasets

Dataset 1 - Parkinson's Disease Classification

Dataset is shared on archive.ics.uc.edu via this link and collected by Departament of Neurology in Cerapasa, Istanbul University. It's target is a binary flag indicating whether if patient suffers Parkinson's Disease (PD). Various medical features were gathered using speech signal processing algorithms that are useful regarding assessment of PD. Originally it contained 757 rows and 754 features, but in later analysis due to collinearity (threshold equal to 0.9) of variables their number has been reduced to 312. Thus 600 rows of the original dataset were loaded resulting in 600×312 shape (task requirement: number of variables should be at least 50% of the number of observations). Dataset is complete, it didn't contain missing values.

Dataset 2 - Swarm Behaviour

Dataset is shared on archive.ics.uc.edu via this link and is achieved from an online survey run by UNSW, Australia. Source contains three datasets relating Swarm Behaviour with classification of 'Flocking - Not Flocking', 'Aligned - Not Aligned', and 'Grouped - Not Grouped', although 'Grouped - Not Grouped' was chosen with target being a binary flag. Each boid has the following attributes: xm and ym represent its (X, Y) position, while xVeln and yVeln define its velocity vector. The alignment, separation, and cohesion vectors are given by xAm, yAm, xSm, ySm, and xCm, yCm, respectively. Additionally, nACm indicates the number of boids within the alignment and cohesion radius, and nSm represents the number of boids within the separation radius. These attributes are defined for all boids, from 1 to 200. A boid (short for "bird-oid object") is an agent in a flocking simulation based on Boid's Algorithm, which was introduced by Craig Reynolds in 1986. It models the collective behavior of birds, fish schools, or other swarm-like entities using simple rules. Dataset's shape was 24017×2400, but due to collinearity requirements random 2500 rows and 1500 columns have been used. Due to collinearity (threshold equal to 0.9) of variables their number has been reduced to 1401 ($>$ 50% of no. of observations). Dataset is complete, it didn't contain missing values.

Dataset 3 - Wine Quality

Dataset is shared on kaggle.com via this link. It is based on archive.ics.uc.edu dataset, but imbalanced data was dealt using cGAN. The target indicates the quality of the wine, which is a number between 3 and 9. We transformed it to binary flag with threshold 5 (> 5 is one, $<= 5$ is zero). Features represent the parameters of a given wine. The dataset's shape was 21000×12. Thus, we've taken 2000 rows, kept 12 features and added dummy variables 99 times that were copies of the original variables with permuted values. That way result data frame has 2000 rows and 1100 non-collinear features ($>$ 50% of no. of observations) and no missing values.

Dataset 4 - Pumpkin Seeds

Dataset is shared on kaggle.com via this link, which describes the physical characteristics of pumpkin seeds classified into two quality classes - Çerçevelik mapped to value 1 and Ürgüp Sivrisi mapped to 0. The original dataset consisted of 2500 rows and 13 feature columns, including the target column, which was reduced by 20% to 2000 rows in the final dataset. To increase the number of features, an analogical method to the wine dataset was used, and 89 permutations of 12 feature columns were added to the dataset. The final dataset consists of 1080 non-collinear features and 2000 rows. Dataset doesn't contain any missing values.

Synthetic dataset

A synthetic dataset is generated using the following procedure, where p, n, d, g are parameters that vary in the experiments:

1. Generate a binary class variable $Y \in \{0, 1\}$ from a Bernoulli distribution with class prior probability p.
2. Generate a feature vector X such that:
   - $X \mid Y = 0$ follows a d-dimensional multivariate normal distribution with mean vector $(0, \ldots, 0)$ and covariance matrix S defined as:
   $$S[i, j] = g^{|i-j|}. \tag{1}$$
   - $X \mid Y = 1$ follows a d-dimensional multivariate normal distribution with mean vector $\left(1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{d}\right)$ and the same covariance matrix:
   $$S[i, j] = g^{|i-j|}. \tag{2}$$
3. Generate n observations using the above steps.

1.2 LogRegCDD implementation

The implemented algorithm is a logistic regression model designed for using coordinate descent with elastic net regularization. The core idea is to estimate the parameter vector β by iteratively updating each coordinate in a way that minimizes the objective function (simplified formula for $y_i \in \{-1, 1\}$):

$$L(\beta) = \sum_{i=1}^{n} \log(1 + e^{-y_i x_i \beta}) + \lambda \left(\frac{1-\alpha}{2} |\beta|_2^2 + \alpha |\beta|_1 \right), \tag{3}$$

where α controls the balance between L1 and L2 regularization, and λ is the regularization strength.

The probability of $Y = 1$ given X is computed using the sigmoid function:

$$P(Y = 1 | X = x) = \frac{1}{1 + e^{-x\beta}}. \tag{4}$$

Each coordinate β_j is updated iteratively using a soft-thresholding approach:

$$\beta_j \leftarrow \frac{S(\sum_i w_i x_{ij} (z_i - (x_i^{(-j)})^T \beta^{(-j)}), \alpha\lambda)}{\sum_i w_i x_{ij}^2 + \lambda(1-\alpha)}, \tag{5}$$

where $S(x, \gamma)$ is the soft-thresholding operator applied to encourage sparsity.

$$S(x, \gamma) = \begin{cases} x - \gamma, & \text{if } x > \gamma \\ 0, & \text{if } |x| \leq \gamma \\ x + \gamma, & \text{if } x < -\gamma \end{cases} \tag{6}$$

$$z_i = x^\top \beta + \frac{y_i - p(x_i)}{w(x_i)} \tag{7}$$

$$w(x) = p(x) \cdot (1 - p(x)) \tag{8}$$

The CCD method updates the model parameters sequentially in coordinate descent manner by optimizing one coordinate at a time while keeping others fixed.

In out implementation, the input matrix is standardized first. By default, we center each column and then scale them. It is also possible not to center data in case of a sparse input matrix.

The methods operate internally with β vector for normalized data. Intercept is always added to retain information about apriori distribution of target variable. For external access, rescaled β for original data is returned. β is initialized with zeros.

The class enables computing λ_{max} value. Of course, only values $\lambda \leq \lambda_{max}$ should be used. Every time a `fit` method is applied, we create a set of active coordinates and put all the coordinates into it. The method iterates only over elements from the set. Whenever a coordinate is zeroed, it is removed from the set. When applying decreasing λ path, `fit` method is used several times, for each λ value, so the set is recreated for each value, enabling activating other coefficients.

Early stopping mechanism is also implemented in `fit` method. The training is stopped when

$$\frac{\|\beta^{new} - \beta^{old}\|_2}{\|\beta^{old}\|_2} < \epsilon. \tag{9}$$

The condition is checked after the full cycle over coordinates.

In [1] several ways of optimizing the algorithm were proposed. However, they are dedicated for Iterated Least Squares method and they are not directly adaptable to the weighted version. To apply them, the weights need to be fixed, at least for a full cycle. This leads to inaccurate hessian matrix estimation in Newton-Raphson method and may decrease the convergence ratio. According to this, no other optimizations were implemented.

2 Correctness of the LogRegCDD algorithm

2.1 Performance of the algorithm at $\lambda = 0$

When setting $\lambda = 0$, no regularization is added. For simple linearly separability case 1, the coefficients values were very high: -51.93, 8.80 and 11.06.

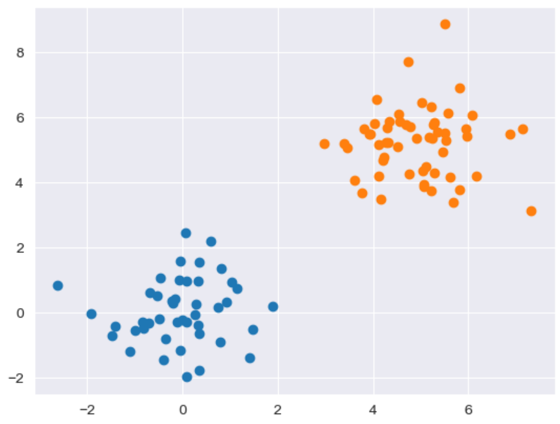


Figure 1: Linearly separable points

We compared also how our custom model works with and without regularization. The models were trained for 20 iterations on Parkinson's data set. For regularized model, α and λ were set to 0.9 and 1 respectively. The training for the regularized model was faster due to updating only active coefficients. It took less than 4 second. For the other it was almost 8 seconds. Additionally, regularized version's accuracy was higher. The results for the models are: 0.833 and 0.817.

2.2 Likelihood function values and coefficient values depending on iteration

We trained our model on Parkinson's data set for 100 iterations, with $\alpha = 0.9$ and $\lambda = 0.01$. We measured how log-likelihood and coefficients' values change over iterations [2, 3]. It is visible, that after some iterations the model stabilizes.

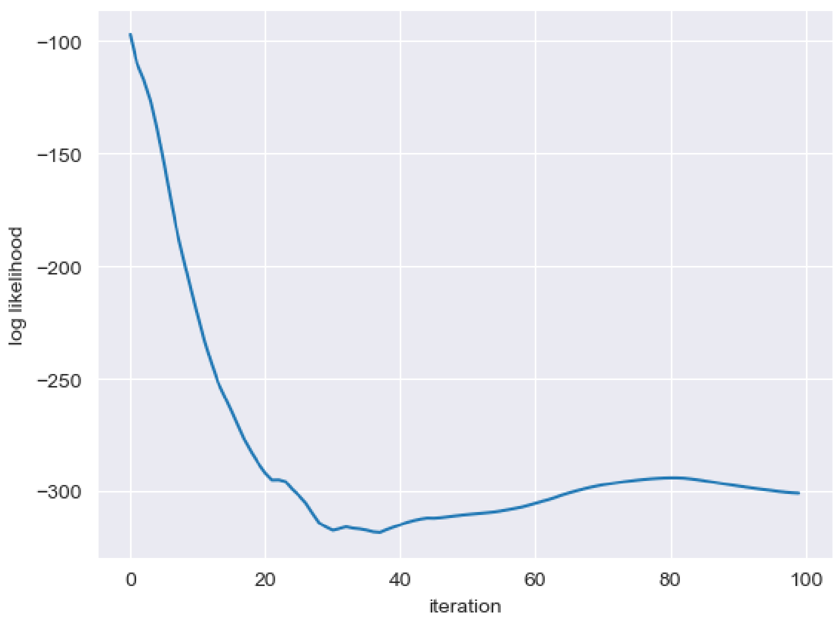


Figure 2: Log-likelihood values over iterations

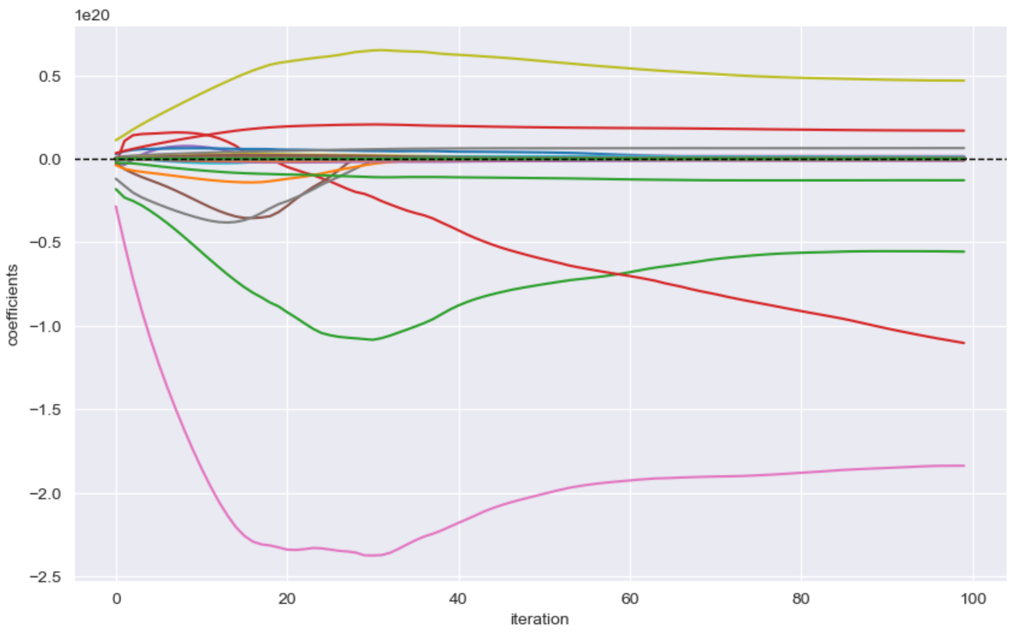


Figure 3: Values of coefficients over iterations

2.3 Comparison with ready implementation of logistic regression with L1 penalty

We compared our model with the one from sklearn package using only L1 penalty. λ was set to 1 in both cases. The custom model was trained for 100 iterations. Testing results are stated in the table 1. The obtained scores are very similar to each other in terms of all analyzed measures. Both models selected almost the same number of features: 312 for our model and 313 for sklearn one.

Metric	custom model	sklearn model
Accuracy	0.7416	0.7412
ROC AUC	0.8342	0.8543
F1 Score	0.8881	0.8791
Precision	0.9033	0.9191
Recall	0.5849	0.5900
AUC	0.9229	0.9434

Table 1: Comparison of custom and sklearn models with L1 penalty

3 Evaluation of the LogRegCDD algorithm

Below impact of dataset parameters: n.p,d,g on the performance of LogRegCCD algorithm are described. For reference, these are the meanings of them:

- $p \in [0, 1]$ - bernoulli distibution probability of class 1
- $n \geq 1$ - number of samples
- $d \geq 1$ - number of features
- $g \in [0, 1]$ - covariance decay factor (controls how strongly features are correlated in the covariance matrix)

In the experiments, the LogRegCDD algorithm performed with a fixed number of iterations equal to 10 and α, λ equal to 0.9. Together, grid search of a total of 108 combinations was computed. Below are shown values and conclusions from results regarding parameters' impact. In tables, the first column relates to the parameter, other 4 to balanced accuracy (threshold 0.5) and area under the ROC curve measures for LogRegCDD algorithm and sklearn's LogisticRegression (no penalty) for reference.

p	**test_acc_ccd**	**test_acc_sklearn**	**test_roc_ccd**	**test_roc_sklearn**
0.1	0.54	0.55	0.66	0.6
0.25	0.61	0.6	0.69	0.64
0.5	0.64	0.59	0.69	0.63

Table 2: Comparison of parameter: p

As p increases and the dataset becomes more balanced, LogRegCCD's performance improves, achieving higher accuracy and AUC scores. It consistently outperforms sklearn's Logistic Regression, demonstrating robustness to class imbalance.

n	**test_acc_ccd**	**test_acc_sklearn**	**test_roc_ccd**	**test_roc_sklearn**
100.0	0.58	0.54	0.64	0.56
500.0	0.6	0.59	0.69	0.64
1000.0	0.62	0.61	0.71	0.67

Table 3: Comparison of parameter: n

A larger dataset leads to better performance, with both accuracy and AUC increasing as n grows. LogRegCCD benefits from more data, showing stronger generalization than sklearn's model, especially for small sample sizes

d	**test_acc_ccd**	**test_acc_sklearn**	**test_roc_ccd**	**test_roc_sklearn**
50.0	0.64	0.64	0.75	0.72
100.0	0.62	0.6	0.7	0.65
500.0	0.57	0.53	0.63	0.56
2000.0	0.57	0.55	0.63	0.56

Table 4: Comparison of parameter: d

Performance declines as d increases, with the best results observed for lower feature dimensions. High-dimensional data negatively impacts accuracy, likely due to the curse of dimensionality, reducing model effectiveness.

g	**test_acc_ccd**	**test_acc_sklearn**	**test_roc_ccd**	**test_roc_sklearn**
0.1	0.6	0.57	0.68	0.61
0.5	0.59	0.57	0.66	0.6
0.9	0.61	0.6	0.7	0.66

Table 5: Comparison of parameter: g

Higher feature correlation (g closer to 1) slightly improves accuracy and AUC, indicating that LogRegCCD benefits from some level of feature dependence. However, the effect is less pronounced compared to other parameters.

n	**d**	**test_acc_ccd**	**test_acc_sklearn**	**test_roc_ccd**	**test_roc_sklearn**
100.0	50.0	0.66	0.66	0.77	0.7
100.0	100.0	0.6	0.54	0.65	0.55
100.0	500.0	0.52	0.46	0.57	0.46
100.0	2000.0	0.52	0.52	0.56	0.54
500.0	50.0	0.62	0.64	0.73	0.72
500.0	100.0	0.62	0.62	0.7	0.66
500.0	500.0	0.58	0.54	0.66	0.61
500.0	2000.0	0.59	0.56	0.66	0.56
1000.0	50.0	0.63	0.63	0.76	0.75
1000.0	100.0	0.64	0.64	0.74	0.73
1000.0	500.0	0.62	0.59	0.67	0.63
1000.0	2000.0	0.58	0.57	0.67	0.59

Table 6: Relation between n and d parameters

LogRegCCD performs best when d < n, where sufficient samples enable better generalization. When d $\approx$ n, accuracy and AUC decline, and when d > n, performance drops significantly due to the curse of dimensionality. Increasing n mitigates this effect but does not fully counteract high d. Compared to sklearn's unregularized Logistic Regression, LogRegCCD consistently performs better, especially in high-dimensional settings, suggesting that its built-in regularization helps manage feature redundancy and overfitting.

4 LogRegCDD vs sklearn's LogistictRegression

The below tables present a comparison of 2 algorithms: custom-built LogRegCDD (cdd) and sklearn's LogisticRegression (sklearn) without regularization. Values indicate measures of balanced accuracy (acc), ROC AUC (roc), f1-score (f1) and Recall-Precision AUC (rp) on real datasets.

dataset	test_acc_ccd	test_acc_sklearn	test_roc_ccd	test_roc_sklearn
parkinson	0.792	0.799	0.9	0.877
swarm	0.999	1.0	1.0	1.0
wine	0.484	0.485	0.475	0.48
pumpkin	0.795	0.762	0.865	0.836

Table 7: Measures of LogRegCDD vs LogisticRegression - set 1

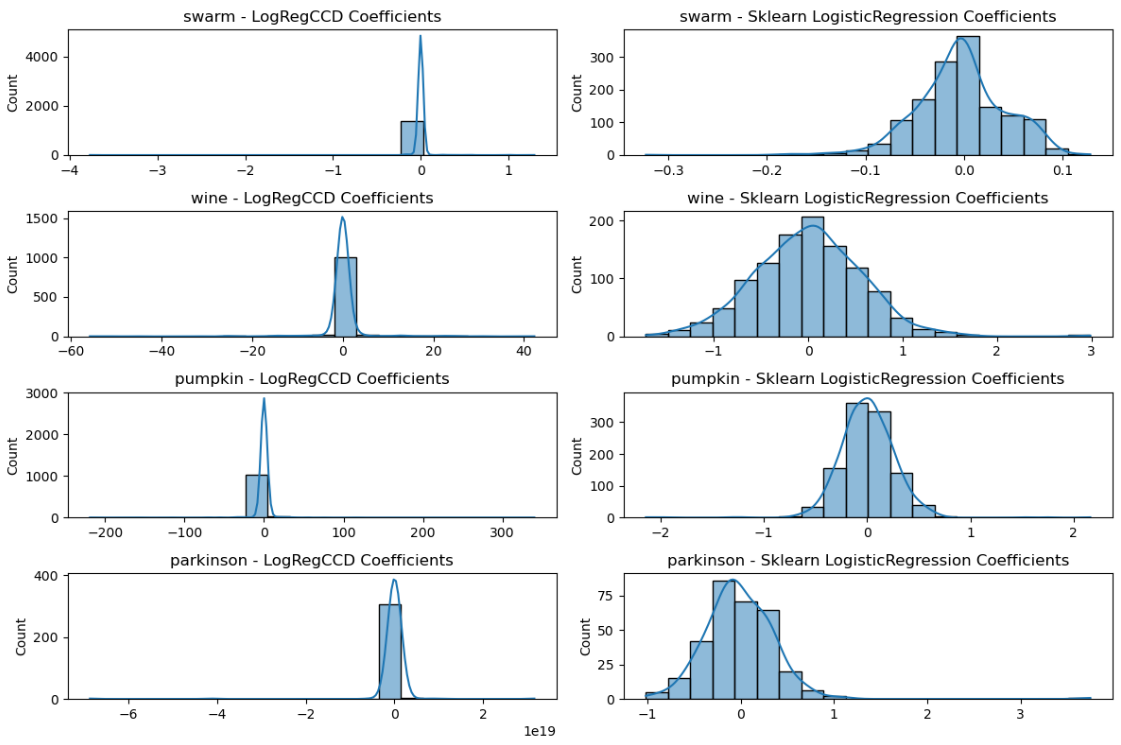
dataset	test_f1_cdd	test_f1_sklearn	test_rp_cdd	test_rp_sklearn
parkinson	0.9	0.9	0.959	0.952
swarm	0.998	1.0	1.0	1.0
wine	0.553	0.558	0.548	0.559
pumpkin	0.8	0.767	0.852	0.822

Table 8: Measures of LogRegCDD vs LogisticRegression - set 2

Performance comparison between the custom LogRegCCD function and scikit-learn's LogisticRegression across four datasets — Parkinson, Swarm, Wine and Pumpkin — reveals notable differences. For the Parkinson dataset, both models achieve similar balanced accuracy, with LogRegCCD scoring 0.792 and sklearn 0.799. The ROC AUC is slightly higher for LogRegCCD (0.899) than sklearn (0.877), while the F1 scores are identical. The Recall-Precision AUC (RP AUC) follows a similar trend, with LogRegCCD achieving 0.959 and sklearn 0.952. In the Swarm dataset, both models achieve near-perfect performance across all metrics, indicating an easily separable dataset (RP AUC = 1.0 for both models). However, for the Wine dataset, both models struggle, with balanced accuracy around 0.484 and lower ROC AUC scores (0.475 for LogRegCCD, 0.480 for sklearn). Interestingly, sklearn's Logistic Regression slightly outperforms LogRegCCD in terms of both F1 score (0.558 vs. 0.553) and RP AUC (0.559 vs. 0.548), suggesting that the classification performance is marginally better in this case. The Opposite happened with LogRegCCD outperforming LogisticRegression on the Pumpkin dataset across all metrics. Considerable differences are between such metrics as accuracy (0.795 vs. 0.762) and F1 (0.8 vs. 0.767), which are closer in value in other datasets. With exceptions to the last dataset, both models show comparable results, with minor variations across datasets.

The comparison between the custom LogRegCCD function, which includes regularization, and scikit-learn's Logistic Regression, which is used without regularization, provides insights into the impact of regularization on model performance. In the Parkinson dataset, both models perform similarly, though LogRegCCD achieves a slightly higher ROC AUC (0.899 vs. 0.877), suggesting that regularization may contribute to better generalization. The RP AUC values (0.959 vs. 0.952) further reinforce this observation. In the Swarm dataset, both models achieve near-perfect scores, indicating that regularization has little impact when the dataset is easily separable. However, in the Wine dataset, where the classification task is more challenging, the lack of regularization in LogisticRegression results in a marginally better F1 score (0.558 vs. 0.553) and RP AUC (0.559 vs. 0.548), possibly due to reduced bias. The benefits of regularization are visible in the case of the Pumpkin dataset, which is more challenging than the Swarm and Parkinson datasets. Still, the metrics are better than the ones for the Wine dataset. This suggests that regularization in LogRegCCD might be introducing some bias that limits flexibility on complex datasets. Overall, regularization in LogRegCCD appears to be beneficial in preventing overfitting on well-structured datasets like Parkinson but might slightly restrict performance in datasets where more flexibility is needed.

Below are presented coefficient values obtained in LogRegCCD ($\lambda, \alpha = 0$) and sklearn's Logistic Regression.



The analysis of the histograms reveals distinct differences between the two logistic regression algorithms across the three datasets. For LogRegCDD (regularized logistic regression), the coefficients are mostly zero for all datasets, with the non-zero values ranging from relatively narrow to wide spans, such as -7 to 3 for Parkinson, -55 to 45 for Wine, -4 to 1.5 for Swarm and -200 to 300 for Pumpkin. This sparsity indicates that regularization with L1 and L2 penalties effectively drives many coefficients to zero, suggesting that only a few features are deemed significant by the model.

In contrast, Sklearn Logistic Regression without penalties produces coefficients that resemble a normal distribution, typically ranging from -1 to 1 for Parkinson, -2 to 2 for Wine, -0.12 to 0.12 for Swarm and -0.5 to 0.5 for Pumpkin, though extreme values can occasionally appear, reaching up to 4, 3, -0.3 and 2, respectively. These distributions indicate that the model is more flexible, allowing coefficients to take a wider range of values and not forcing sparsity. This flexibility may result in overfitting, as the model fits more closely to the data without any regularization constraints.

Overall, the regularized logistic regression approach with LogRegCDD tends to favor simplicity and sparsity, focusing on fewer features, while the unpenalized Sklearn Logistic Regression offers more flexibility but may risk overfitting by not restricting the coefficient values.

References

[1] Jerome H. Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.