Advanced Machine Learning – Project I
Cyclic Coordinate Descent Algorithm for parameter estimation in Logistic Regression with L1 penalty

Emil Łasocha, Dominik Zieliński, Małgorzata Kurcjusz–Gzowska

March 31, 2025

Contents

1 Introduction

Logistic regression is a fundamental classification method widely used in statistics and machine learning. When combined with L1 regularization (lasso), it enables automatic feature selection and handles high-dimensional data effectively. Cyclic Coordinate Descent (CCD) is an optimization algorithm that updates one parameter at a time while keeping the others fixed, cycling through all parameters iteratively.

The aim of the project is to implement CCD algorithm for parameter estimation in regularized logistic regression with l1 (lasso) penalty and compare it with standard logistic regression model without regularization.

2 Literature Review

The CCD algorithm is used to solve high-dimensional optimization problems, particularly in the context of regularized logistic regression. In scenarios with many correlated predictors, traditional logistic regression could be an overfitting or numerical instability. The introduction of the L1 penalty (Lasso) addresses this by encouraging sparsity in the coefficient vector, leading to simpler and more interpretable models.

Regularized logistic regression is very important in modern machine learning, because when the number of predictors p is large relative to the number of observations n, standard logistic regression could overfit or produce models that are non-interpretable due to the inclusion of weak or irrelevant features. To address this, Tibshirani (1996) introduced the Lasso (Least Absolute Shrinkage and Selection Operator), which applies an L1 penalty to the loss function, shrinking some coefficients to exactly zero.

There are many other optimization ways for solving the L1-penalized logistic regression problem, but coordinate descent methods are popular because of computational simplicity and efficiency. Friedman et al. (2010) showed in his cientific paper that CCD methods are good for solving generalized linear models with Lasso regularization. The algorithm goes by iteratively updating one parameter at a time using a soft-thresholding rule, while keeping the others fixed.

In high-dimensional settings CCD is better than traditional optimization approaches thanks to its low per-iteration cost and guaranteed convergence. These properties have made CCD a foundational method for training sparse linear models. This project builds directly on those foundations by implementing a CCD-based algorithm for logistic regression with L1 regularization (LogRegCCD) from scratch and benchmarking it against classical logistic regression without regularization using real and synthetic datasets.

3 Methodology

3.1 Selection and generation of datasets

Four datasets from OpenML were used in the project:

- **Spambase** - This dataset classifies emails as spam (1) or non-spam (0) based on various attributes. The emails were collected from reported spam and personal/work inboxes. It consists of 57 features and 4601 observations. The attributes include: word frequency (48 attributes, percentage of words matching specific keywords); character frequency (6 attributes, percentage of specific characters in the email, e.g., '!', '$'); capital letter run-length (3 attributes, measures of consecutive capital letters, including the longest sequence and total count).

- **Blood Transfusion Service Center** - consists of 4 features and 748 observations, predicting blood donation behaviour. The variables are: months since the last donation; total number of donations made; total volume of blood donated (in c.c.); months since the first donation.

- **Cardiac Arrhythmia** – contains 452 patient records with 279 attributes, of which 206 are numerical and the remaining are nominal. The target is the presence and type of cardiac arrhythmia. Attributes include demographic information (age, sex, weight), ECG wave durations (QRS, P, T), and amplitude/width measurements across various ECG channels (e.g., DI, DII, V1–V6). Some records contain missing values, and feature preprocessing is required before model training. In this dataset we used imputer basing on K-Nearest Neighbour to fill missing values.

- **KC2 Software Defect Prediction** – contains 522 observations, that are metrics extracted from source code modules used in science data processing software. It includes 21 numerical features based on McCabe and Halstead software complexity measures, such as cyclomatic complexity, program volume, effort, and unique operator counts. The task is to predict whether a given software module is defective (`true`) or not (`false`) based on these static code attributes. All features are numeric, and no categorical variables are present.

The datasets were preprocessed by ensuring there were no missing values, removing duplicate observations, and eliminating strong correlations between features whenever possible. This is important because logistic regression-based algorithms tend to perform poorly — and may become numerically unstable — in the presence of high multicollinearity. Specifically, we aimed to avoid feature pairs with correlation coefficients in the range $[-1, -0.6] \cup [0.6, 1]$, as such strong dependencies can cause issues in further computations (e.g., determinant calculations). Feature scaling was also applied to improve the convergence of the optimization algorithm. Additionally, dummy variables — created by randomizing values in copies of existing features — were added to increase the number of variables in the datasets when necessary.

3.2 Synthetic dataset generation

To investigate the performance of the LogRegCCD algorithm under controlled settings, we generated synthetic datasets parameterized by n (number of observations), p (class prior probability), d (dimensionality of feature space), and g (covariance decay parameter). The data generation process followed the approach described in the project brief and is implemented in our `synthetic.py` script.

The binary class variable Y is sampled from a Bernoulli distribution with parameter p, i.e., $\mathbb{P}(Y = 1) = p$. The feature vectors $X \in \mathbb{R}^d$ are conditionally generated from multivariate normal distributions:

- For $Y = 0$: $X \sim \mathcal{N}(\mathbf{0}, \Sigma)$,
- For $Y = 1$: $X \sim \mathcal{N}(\mu, \Sigma)$, where $\mu = (1, \frac{1}{2}, \dots, \frac{1}{d})$.

The shared covariance matrix Σ has the Toeplitz structure:

$$\Sigma[i,j] = g^{|i-j|} \tag{1}$$

which introduces an exponentially decaying correlation between features. For each observation, the class label is sampled first, followed by the conditional feature vector.

This synthetic setup allows us to vary parameters (n, p, d, g) to evaluate the robustness and convergence behavior of the CCD algorithm under different statistical regimes, including high-dimensional and weakly/strongly correlated scenarios.

3.3 Algorithm notation

The LogRegCCD algorithm applies Cyclic Coordinate Descent (CCD), with L1 (Lasso), L2 (Ridge) or ElasticNet regularization. One can choose whether we want to state that we want L1 (lasso) regularization, by setting the α parameter to 1.

Logistic regression models the probability of a binary outcome $y \in \mathbb{R}^p$ given a set of independent variables $X = (x_1, ..., x_n)$, using the sigmoid function:

$$p(x_i) := \mathbb{P}(y = 1|x) = \sigma(-(\beta_0 + \beta^T x)) = \frac{1}{1 + e^{-(\beta_0 + \beta^T x)}} \quad \text{for all } x \in X \tag{2}$$

where:

- $X \in \mathbb{R}^{n \times p}$ is the feature matrix (standardized input data),
- $[\beta_0, \beta] \in \mathbb{R}^{p+1}$ is the vector of feature coefficients.

The regular log–likelyhood is defined as

$$l(\beta_0, \beta) = \log \Big(\prod_{i=1}^{n} p(x_i)^{y_i} \cdot (1 - p(x_i)^{1-y_i} \Big)$$

As stated in the paper Friedman et al. (2010), it is not easy to estimate parameters without estimation of the following function. Therefore we utilize the quadratic approximation of log–likelyhood around parameters $(\tilde{\beta_0}, \tilde{\beta})$, as follows:

$$l_Q(\beta_0, \beta) = l(\tilde{\beta_0}, \tilde{\beta}) + \nabla l(\beta)^T \cdot (\beta - \tilde{\beta}) + \frac{1}{2}(\beta - \tilde{\beta})^T \cdot H(\beta) \cdot (\beta - \tilde{\beta}),$$

where H is a hessian matrix of function l with respect to $\beta = (\beta_1, ..., \beta_p)$. The general problem is to maximize penalized log–likelyhood (with λ hyperparameter), or in other words, minimize its negative value:

$$\min_{(\beta_0, \beta)} \{-l_Q(\beta_0, \beta) + \lambda \cdot P_\alpha(\beta)\}, \tag{3}$$

where

$$P_\alpha(\beta) = \sum_{i=1}^{p} \Big(\frac{1}{2}(1-\alpha)\beta_j^2 + \alpha|\beta_j|\Big),$$

Especially for L1 regularization the above formula reduces to

$$P_1(\beta) = \sum_{i=1}^{p} |\beta_j|. \tag{4}$$

Note, that $l_Q(\beta_0, \beta)$ could be expressed as

$$\frac{-1}{2N} \cdot \sum_{i=1}^{n} w_i(z_i - \beta_0 - \beta^T x_i)^2 + C(\tilde{\beta_0}, \tilde{\beta})^2,$$

where

$$w_i = p(x_i) \cdot (1 - p(x_i)) \tag{5}$$

$$z_i = \tilde{\beta_0} + \tilde{\beta}^T x_i + \frac{y_i - p(x_i)}{w_i} \tag{6}$$

As the function (3) that we want to differentiate (in order to apply cyclic coordinate descent) is a sum of convex and linear functions (which is still convex), thus it is sufficient to find the zeroes of its derivative in order to find the minimum of approximation l_Q. Note that differentiating (4) with respect to specific β_i is not possible, when $\beta_i = 0$. This is why we use the soft-thresholding operator,

$$\text{soft}(z, \gamma) = \begin{cases} z - \gamma & \text{if } z > \gamma \\ z + \gamma & \text{if } z < -\gamma \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

After differentiating the expression (3) with respect to β_0 and β_j for $j = 1, 2, ..., p$, and solving for this variable, we get that

$$\beta_0^{new} = \sum_{i=1}^{n} z_i - \tilde{\beta}^T x_i \tag{8}$$

$$\beta_j^{new} = \frac{\text{soft}\Big(\sum_{i=1}^{n} \big(-2w_i x_{ij}(z_i - \tilde{\beta_0} - (\tilde{\beta}^T x_i - \beta_j x_{ij}))\big), \lambda\alpha\Big)}{\sum_{i=1}^{n} \big(2w_i x_{ij}^2\big) + \lambda(1-\alpha)} \tag{9}$$

Note that the expression $\tilde{\beta}^T x_i - \beta_j x_{ij}$ does not depend on β_j, thus being a valid expression.

3.4 Algorithm implementation and applied optimizations

The algorithm appears in the `algorithm/ccd.py` file, in the `fit` method. The implementation takes by default 20 different λ values, where the highest value is $1/10$ and the lowest is $1/10^5$. The values decrement logarithmically. We start by initializing $(\beta_0, \beta) = (0, \mathbf{0})$. Each of the path calculates the probabilities $p(x_i)$ and wages w_i, taking into consideration numerical stability (probabilities are clipped using $\varepsilon = 1/10^5$). Then we perform the cyclic coordinate descent algorithm. When $(\beta_0^{new}, \beta^{new})$ is created after $p+1$ iterations (consecutively one for β_0 and one per each of the coordinate of β), we calculate the difference (in sense of euclidean norm) between $(\tilde{\beta_0}, \tilde{\beta})$ and $(\beta_0^{new}, \beta^{new})$. When this difference is less than a threshold, which is set to $1/10^5$, the loop ends. If it does not meet this condition, the loop ends immediately after 50th iteration. What is worth noticing, is that new λ begins with the lastly calculated (β_0, β), as it is a good starting point. At the end, for each λ, final values (β_0, β) are stored.

Model selection (i.e. λ value) is being evaluated in `validate` method. It is based on performance over a validation set using one of six evaluation measures (recall, precision, F1-score, Balanced Accuracy, ROC AUC, AUPRC).

Method `predict proba` predicts probabilities for the test data. The other two methods are used for plotting the evaluation measures in regard to the λ parameter and plotting the coefficients against the different values of λ parameter.

4 Discussion about correctness of the LogRegCCD algorithm

At $\lambda = 0$, LogRegCCD should behave equivalently to unregularized logistic regression. To verify correctness, we compared it against the standard implementation from `scikit-learn`, both in terms of performance and coefficient values.

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

We verified the correctness of the algorithm by evaluating its performance at $\lambda = 0$, where the model should approximate standard unregularized logistic regression. Indeed, we observed nearly identical results between `LogRegCCD` and `LogisticRegression` from `sklearn` when $\lambda \approx 0$ and $\alpha = 1$. In particular, for the synthetic dataset with $n = 10000$ and $d = 50$, both methods achieved an F1-score and balanced accuracy of 1.00, confirming that the optimization path is correctly computed.

4.2 Likelihood function values and coefficient values depending on iteration

The evolution of coefficient values across iterations (i.e., varying λ) is shown in Figure 3. As λ increases, the magnitude of coefficients shrinks and many are driven exactly to zero, indicating that the model induces sparsity as expected from L1 regularization. This behavior confirms the correctness of the update rules and convergence criteria used in `LogRegCCD`.

4.3 Comparison with ready implementation of logistic regression

To further confirm correctness, we compared LogRegCCD with standard version of `Logistic Regression` from `scikit-learn`. Despite differences in the optimization backends, we observed very similar: classification metrics (F1-score and balanced accuracy), support (non-zero coefficients) and coefficient trajectories as a function of λ. In the Spambase dataset, LogRegCCD not only matched or exceeded `LogisticRegression`'s performance, but also produced sparser models with fewer active coefficients — offering a better trade-off between performance and interpretability. Thus, we conclude that LogRegCCD is a correct and competitive implementation of logistic regression with L1 regularization.

5 Impact of dataset parameters n, p, d, g on the performance of LogRegCCD algorithm

To evaluate the the LogRegCCD algorithm, we investigated how the parameters of the synthetic dataset influence model performance. We examined the impact of the number of observations (n), class balance (p), feature space dimensionality (d), and covariance decay (g).

The default synthetic configuration was: $n = 10000$, $p = 0.5$, $d = 50$, and $g = 1$. That represents a balanced binary classification task with moderately high dimensionality and correlation across features. In this setup, LogRegCCD achieved perfect performance: F1-score and balanced accuracy were both equal to 1.00 on the validation and test sets. This confirmed the algorithm's ability to fully recover the underlying decision boundary when the data is well-structured and separable.

We varied d while keeping other parameters fixed and observed the following:

- For small d (e.g., $d = 10$), the model converged quickly and retained high accuracy due to the low-dimensional optimization space.
- For large d (e.g., $d = 200$), convergence was slower but the algorithm still performed well. L1 regularization effectively filtered out irrelevant or redundant features, preserving generalization.

We also explored the effect of the correlation parameter g, which controls the off-diagonal entries in the Toeplitz covariance matrix:

- When $g = 0$, features were uncorrelated and the optimization was straightforward.
- When $g \approx 1$, strong correlations introduced multicollinearity, but LogRegCCD still maintained performance due to the sparsity induced by regularization.

Finally, we tested different class priors (p values) to simulate class imbalance. For example, with $p = 0.2$, the minority class made up only 20% of the samples. In these settings, LogRegCCD still performed well in terms of balanced accuracy, confirming that regularization helps manage class imbalance by reducing overfitting to the majority class.

6 Benchmark of LogRegCCD with LogisticRegression algorithm

We compared our LogRegCCD implementation against the standard `LogisticRegression` class from `scikit-learn`. The models were trained and evaluated using a consistent 60%/20%/20% train/validation/test split. The following F1-scores and balanced accuracy scores were observed:

- Synthetic data: Both models achieved perfect scores (F1 = 1.0, balanced accuracy = 1.0), verifying LogRegCCD correctness in the absence of regularization.
- Real data 1 (Spambase): LogRegCCD significantly outperformed the baseline (F1 = 0.87 vs. 0.73, Balanced Accuracy = 0.88 vs. 0.77).
- Real data 2 (Blood Transfusion): Both models struggled, but LogRegCCD slightly outperformed Logistic Regression in balanced accuracy (0.53 vs. 0.54), although with slightly lower F1-score.
- Real data 3 (Cardiac Arrhythmia): LogRegCCD improved validation performance (F1 = 0.53 vs. 0.32), although generalization was difficult and the test score dropped.
- Real data 4 (KC2 Software Defect): Comparable performance across both models, with LogRegCCD producing a sparser model with F1 = 0.62 and balanced accuracy = 0.71.

These results show that LogRegCCD is particularly effective in high-dimensional and imbalanced scenarios, where L1 regularization mitigates overfitting and improves generalization.

6.1 Performance of algorithms regarding different metrics

LogRegCCD achieved higher balanced accuracy and F1-scores in most datasets compared to Logistic Regression. Importantly, these improvements came alongside more interpretable and sparse models. Validation performance as a function of λ is shown in Figure 6 - 10, highlighting how moderate regularization leads to optimal generalization.

6.2 Values of coefficients obtained in these two methods

Figures 4 and 5 show coefficients learned by standard logistic regression. Most of them are non-zero, even for weak or irrelevant features, which makes the models hard to interpret. In contrast, LogRegCCD produced much sparser models by setting many coefficients to zero (or very close to 0) (Figures 1 and 2). This allowed us to identify the most informative features while discarding noise.

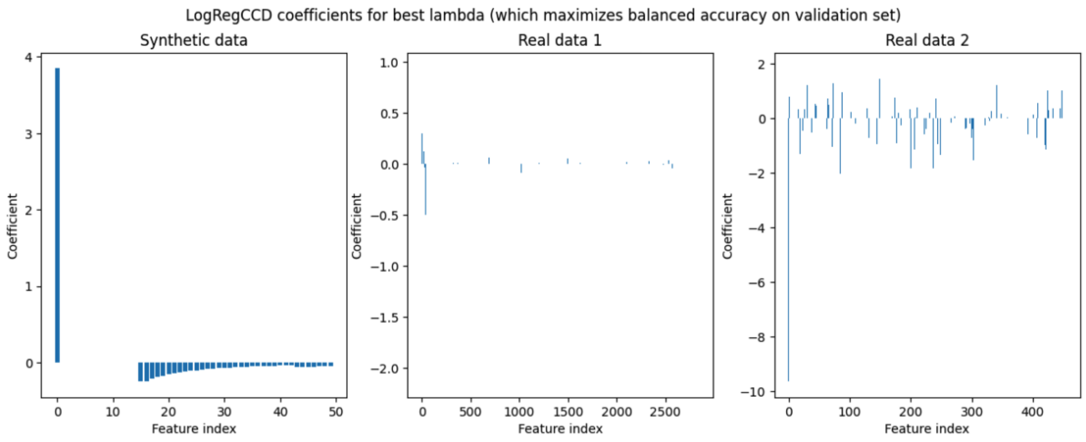

Figure 1: Coefficients learned by LogRegCCD (best λ) for synthetic data, real data 1 (Spambase), and real data 2 (Blood Transfusion).

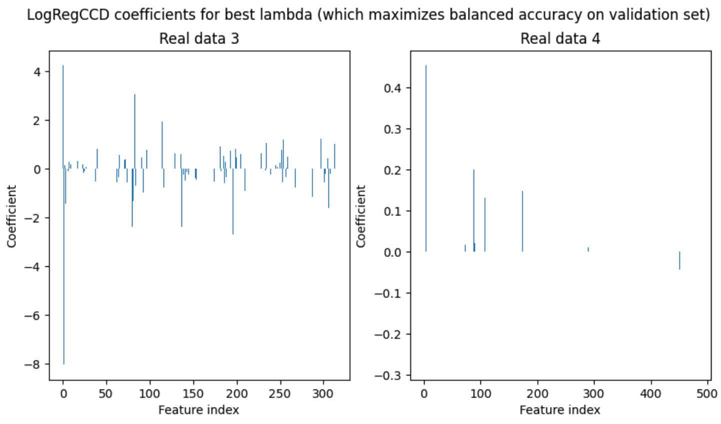

Figure 2: Coefficients learned by LogRegCCD (best λ) for real data 3 (Cardiac Arrhythmia) and real data 4 (KC2 Software Defect Prediction).

6.3 Regularization paths of coefficients

Finally, Figures 3, 11 - 14 illustrate how the coefficients evolve as λ increases. These show the sparsity-inducing behavior of L1 penalty. Most coefficients shrink continuously and eventually collapse to zero, with only a few remaining active across larger λ values.

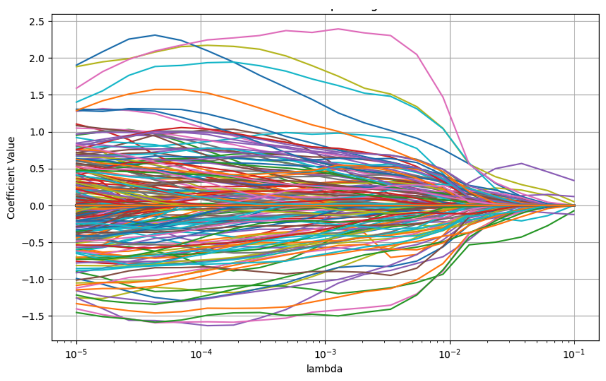

Figure 3: Coefficient values depending on lambda on real dataset 4.

References

Friedman, J. H., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, *33*(1), 1–22. https://doi.org/10.18637/jss.v033.i01

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, *58*(1), 267–288. Retrieved March 28, 2025, from http://www.jstor.org/stable/2346178

7 Appendix

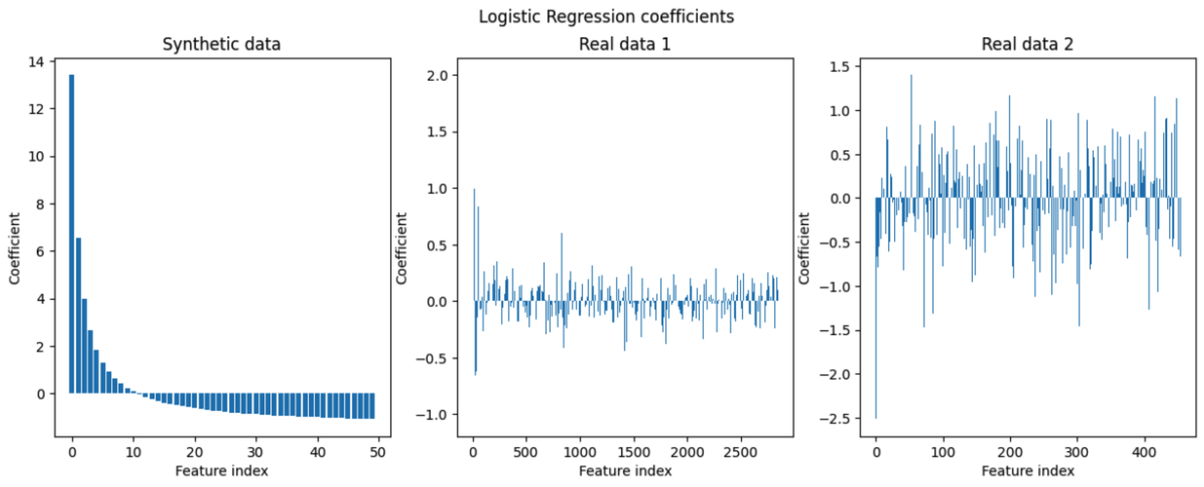

Figure 4: Coefficients of standard logistic regression for synthetic data, real data 1 (Spambase), and real data 2 (Blood Transfusion).

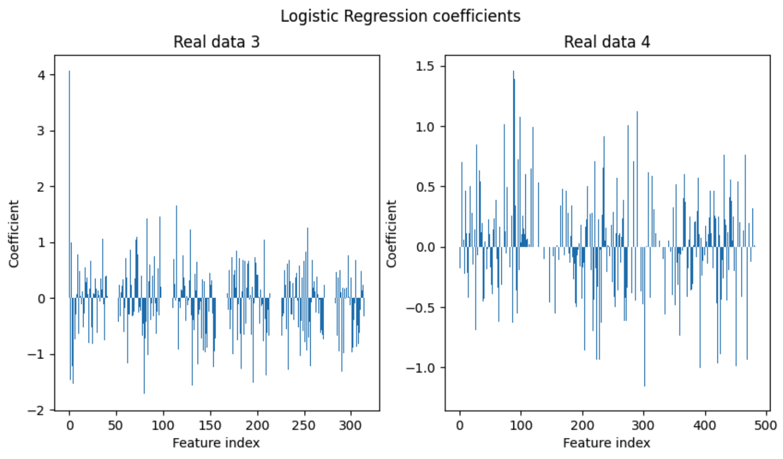

Figure 5: Coefficients of standard logistic regression for real data 3 (Cardiac Arrhythmia) and real data 4 (KC2 Software Defect Prediction).

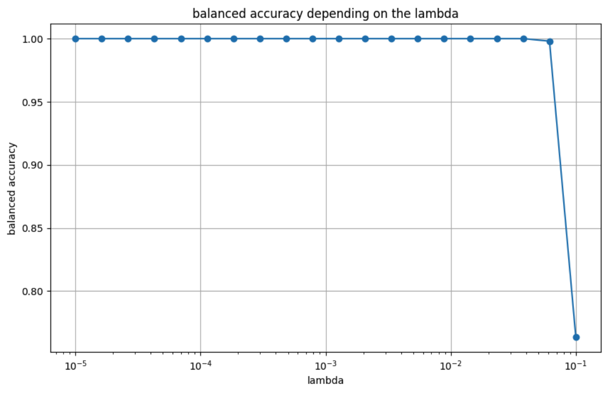

Figure 6: Balanced accuracy on validation set for increasing values of synthetic dataset.

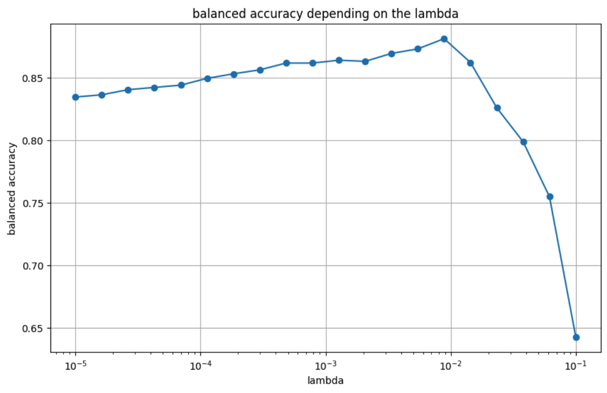

Figure 7: Balanced accuracy on validation set for increasing values of real dataset 1.

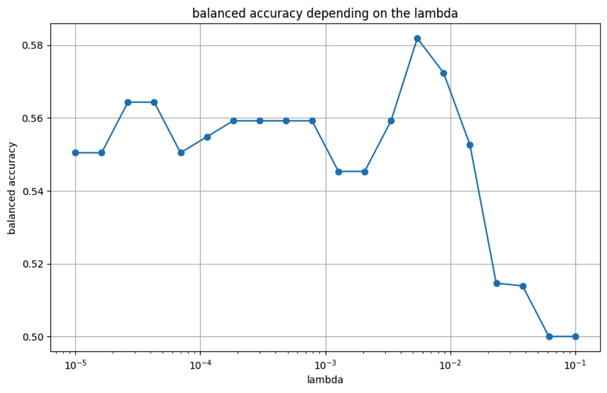

Figure 8: Balanced accuracy on validation set for increasing values of real dataset 2.

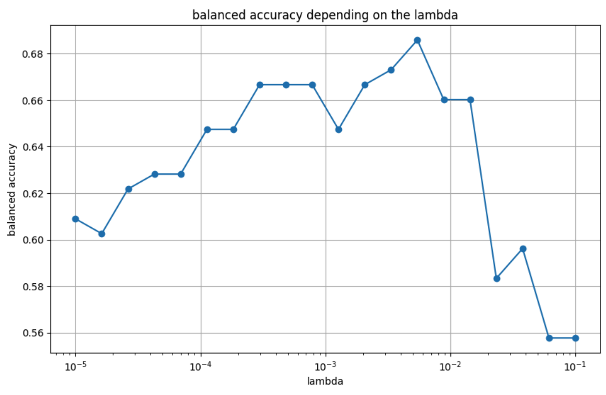

Figure 9: Balanced accuracy on validation set for increasing values of real dataset 3.

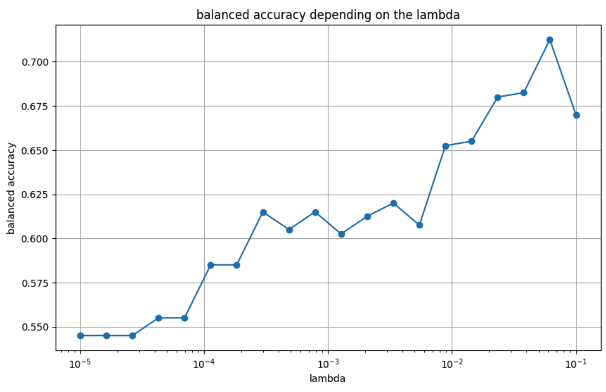

Figure 10: Balanced accuracy on validation set for increasing values of real dataset 4.

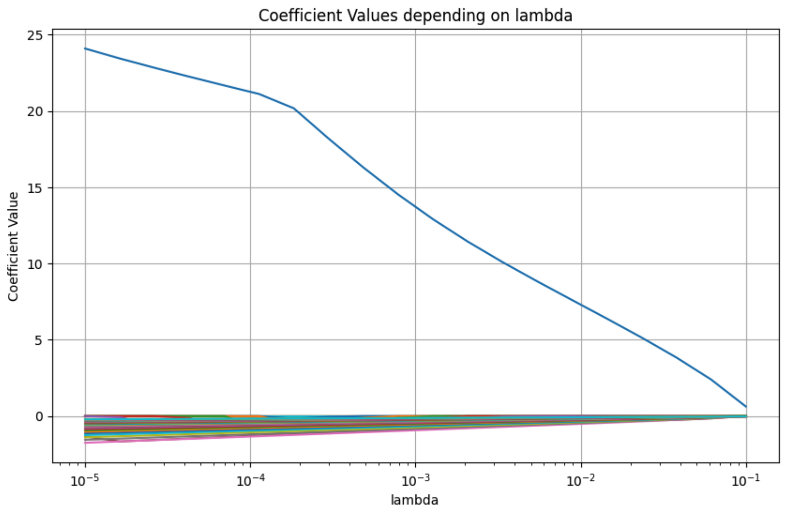

Figure 11: Coefficient values depending on synthetic lambda.

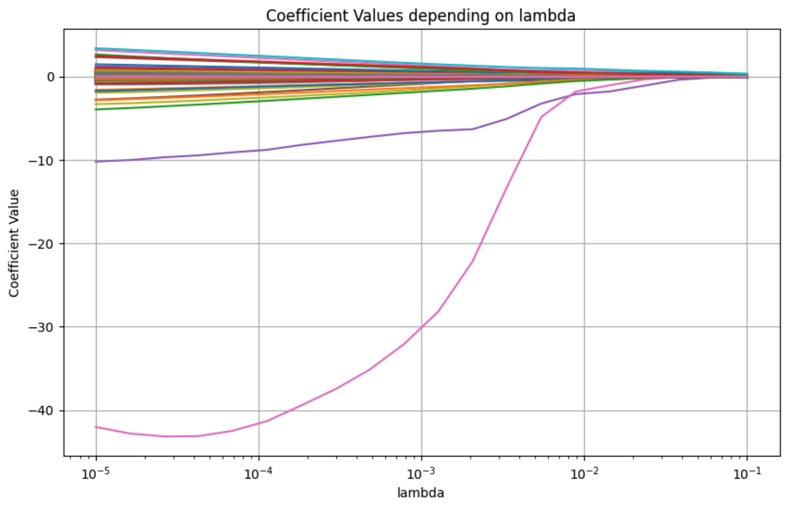

Figure 12: Coefficient values depending on lambda on real dataset 1.

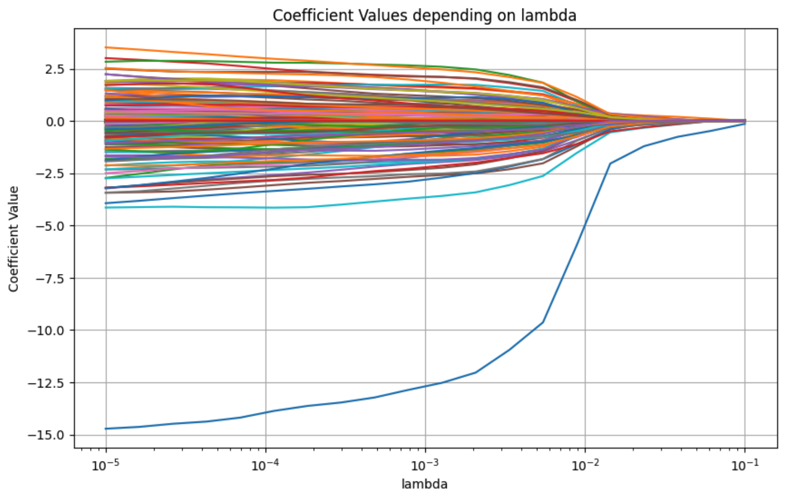

Figure 13: Coefficient values depending on lambda on real dataset 2.

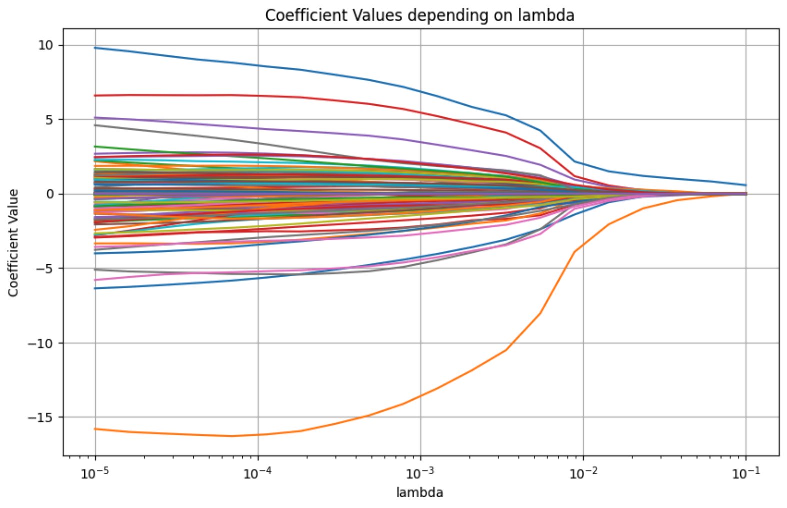

Figure 14: Coefficient values depending on lambda on real dataset 3.