Logistic regression with L1 regularization

Paulina Pacyna
Mateusz Wójcik

April 2021

1 Least angle regression

Least angle regression is an optimization algorithm used in linear regression with $L1$ constraint on the weights. The algorithm implemented by us is based on LARS algorithm decribed in [1]. Lets denote matrix with input variables as X and target variable as y. We aim to estimate β variable such that:

$$y = \beta X + \epsilon, \quad \epsilon \sim N(0, 1)$$

$$||\beta||_{L1} < t$$

The algorithm works as follows:
0. We assume that y has mean equal to 0 and standard deviation equal to 1.
1. Initiate the estimation of y and weights with a zero vector.

$$\hat{y}_0 = \mathbf{0}, \beta_0 = \mathbf{0}$$

2. Compute the correlation of X variable with residual vector:

$$C = X^T(y - \hat{y})$$

3. Let A denote the active set defined as:

$$A = \{i \in \mathbf{N} : \ C_i = \max(C)\}$$

4. Compute sign vector of the correlation:

$$s = [sign(C_i)], \ C_i \in C$$

5. Let X_a be a matrix consisting of columns belonging to X matrix, such that $ainA$ and X_a columns have positive correlation with the residual.

$$X_a = [..., X_a \cdot s_a, ...], \ a \in A$$

6. Compute the Gramm matrix for X_a:

$$G_a = X_a^T X_a$$

7. Compute coefficient:

$$A_a = (\sum_{ij} (G_a^{-1})_{ij})^{-\frac{1}{2}}$$

8. Compute u_a vector. This vector indicates the direction in which we will move our $\hat{y}$ estimation. Let $\mathbf{1}$ be a vector of 1's of length equal to $|A|$

$$u_a = A_a = X_a (G_a)^{-1} \mathbf{1}$$

9. Let $a = X^T u_a$
10. Compute the following scalar:

$$\gamma = \min_{i \in A}^{+} \{ \frac{\max(c) - c_i}{A_a - a_i}, \frac{\max(c) + c_i}{A_a + a_i} \}$$

where $\min^+$ is a minimum taken only over positive values. 11. If γ is close to 0, then stop the algorithm. Otherwise, update the current y estimation:

$$\hat{y}_{k+1} = \hat{y}_k + \gamma u_a$$

12. Compute β_{k+1} from the standard regression formula:

$$\beta_{k+1} = (X^T X)^{-1} X^T \bar{y}$$

13. Repeat steps 2-12.

We can plot $\beta_0 ... \beta_t$ in each step (Figure 1). Different colors represents different variables.

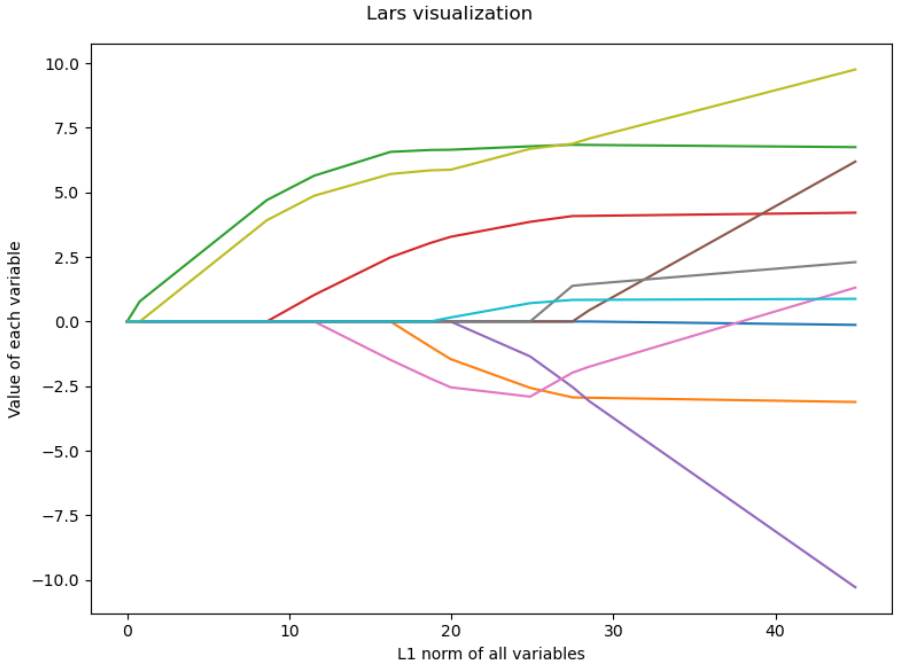

Figure 1: Values of β in each step of LARS algorithm. Different colors represents different variables.

2 IRLS-Lars based

We can extend the algorithm described above to obtain logistic regression. LARS is a starting point in logistic regression algorithm described in [2]. The algorithm reduces penalized logistic regression to penalized linear regression, and uses LARS method to compute the coefficients. Formally, our optimization problem can be written as:

$$\min_{\theta} \sum_{i=1}^{M} -\sigma(-\theta^T x) + C||\theta||_{l1},$$

where σ denotes the sigmoid function, i.e. $\sigma(x) = \frac{1}{1+e^{-x}}$. We can approach this task using Newton's method. If the denote by θ_k the approximation of theta in k-th step, then we can compute the step direction: $\gamma_k = \theta_k - H^{-1}(\theta_k)g(\theta_k)$ H^{-1} is a approximation of the Hessian matrix and g is an approximation of the gradient. Then we apply the step direction as follows:

$$\theta_{k+1} = (1-t)\theta_k + t\gamma_k,$$

where t is found user line search algorithm. Going back to our main algorithm, we want to reduce the problem to linear regression. We start the algorithm with $\theta = \mathbf{0}$. X_i denotes the i-th observation. We define the Λ matrix and z vector as follows:

$$\Lambda_{ii} = \sigma(\theta^T X_i)(1 - \sigma(\theta^T X_i))$$

$$\Lambda_{ij} = 0 \text{ for } i \neq j$$

$$z_i = x_i^T \theta_k + \frac{(1 - \sigma(y_i x_i^T \theta_k))y_i}{\Lambda_{ii}}$$

for $i, j = 1, ..., M$ Next, we want to minimize

$$||\Lambda^{\frac{1}{2}} X^T \gamma - \Lambda^{\frac{1}{2}} z||_2^2$$

subject to $||\gamma||_1 < C$.
This problem can be solved by the LARS algorithm. The solution of this algorithm is denoted as γ and represents our step direction. Our next θ_{k+1} is computed as follows:

$$\theta_{k+1} = t\gamma_k + (1-t)\theta_k$$

where t is chosen such that θ_{k+1} minimizes $\sum_{i=1}^{M} -\sigma(-\theta^T x)$ After the new $theta$ is computed, we repeat the process.

3 Different Optimisation Algorithms and Implementations of L1 regularisation

We have also tested different algorithms of optimizing logistic regression parameters. First, we wanted to test well known algorithms that are usually applied without L1 regularisation in conjuntion with non-differentiable loss function.

- gradient descent
- stochastic gradient descent - implementation with dividing the training data into batches at random and updating the weights for each batch
- newton method

To implement $L1-$regularisation in these algorithms, we tried to test a few different approaches. We first needed to smoothen the cost function which is a sum of absolute values of the weights, $\|w\|_1 = \sum_{i=1}^{n} |w_i|$. To do so, we were thinking of three different approaches.

Let $\epsilon > 0$ be a small number.

- epsilon $L1$ function approximated by a square root of the shifted weight squared
$$\text{epsL1}(w_i, \epsilon) = \sqrt{w_i^2 + \epsilon},$$
where w_i is one of the weights.
- Huber function
$$\begin{cases} w_i^2/(2\epsilon) & \text{if } |w_i| < \epsilon \\ |w_i| - \epsilon/2 & \text{otherwise} \end{cases}.$$
- assuming that the partial derivative of $|w_i|$ with respect to $|w_i|$ for $w_i = 0$ is equal to 0.

All of the methods are listed and used in [2]. Basically, we wanted to use these functions multiplied by the regularisation parameter to calculate the gradient and update the weights (for GD and SGD algorithms). In the case of Newton method, we wanted to use the gradient of the functions and transform it to the diagonal matrix, which then is used to update the hessian that is inversed in the algorithm.

Nonetheless, after consultations with mr. Damian Suski, we decided not to smoothen L1 norm in the calculations. According to the literature and personal experience, smoothening only seemingly solves the problem of non-differentiability. Values of gradients in such cases still drastically oscillate near points where we adjust the cost function. Therefore, we started looking for different implementations of optimizers that we could use to employ for logistic classification model with L1 regularisation. We used following implementations:

- *SGDClassifier* - linear classifier with Stochastic Gradient Descent training from *sklearn* Python module.
- *LogisticRegression* - *sklearn* model used with *liblinear* and *saga* solvers and L1 penalty.
- *glmnet* - package that fits a generalized linear model via penalized maximum likelihood, also Python wrapper for the fortran library used in the R package glmnet limited to linear and logistic regression. Algorithm applies **coordinate descent**.
- *CVXPY* - Python-embedded modeling language for convex optimization problems. It relies on the open source solvers **ECOS**, **OSQP**, and **SCS** which are chosen according to the problem.

3.1 Datasets Used

For experiments, we used a few datasets for binary classification. Main results are presented using the following datasets.

- **Breast Cancer** - dataset describing health state of a patients suffering from breast cancer. The target variable describes if the cancer is maligant. It consists of 30 variables and 569 observations.
- **NBA** - dataset on all the NBA finals results and NBA Regular Seasons MVP. It consists of 12 variables and 1320 observations.
- **Bankruptcy** - this is a dataset from a recruiting task from a consulting company. Variables include information of each candidating company, for example income, marketing statistics, ratings, sector, number of employees, etc. The target variable represents information about bankruptcy. This dataset is significantly larger than others. It consists of 17 variables and 10 000 observations.

During the experiments, each dataset was divided into training and test dataset with 0.2 ratio for test size.

We wanted to focus on different applications of Machine Learning. Therefore, we used the datasets connected to medicine, sports and financial sectors.

3.2 SGDClassifier

This classifier implements regularized linear models with stochastic gradient descent learning [3]. The gradient in each iteration is estimated for the each sample at a time. The model is updated with a decreasing learning rate. SGD also allows learning by using minibatch. The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using the absolute norm L1 in our case.

Let's describe the mathematical details of the SGD procedure [4]. Given a set of training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in R^m$ and $y_i \in \{-1, 1\}$.

In order to make predictions for binary classification, we simply look at the sign of a linear scoring function. To find the parameters, we minimize the regularized training error given by

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w).$$

In our case

$$L(y_i, f(x_i)) = \log(1+\exp(-y_i f(x_i))) \text{ and } R(w) = \sum_{j=1}^{m} |w_j|,$$

where w is a vector of parameters.

Then, a first-order SGD learning routine is implemented. The algorithm iterates over the training samples and for each example updates the model parameters according to the update rule

$$w \leftarrow w - \eta \left[\alpha \frac{\partial R(w)}{\partial w} + \frac{\partial L(w^T x_i + b, y_i}{\partial w}\right],$$

where η is the learning rate. We used *optimal* decay given by

$$\eta^{(t)} = \frac{1}{\alpha(t_0 + t)},$$

where t is the time step and t_0 is determined based on a heuristic proposed by Leon Bottou.

For L1 regularisation the truncated gradient algorithm proposed in [5] is used. It calculates the partial derivative of absolute values as sign $\left(w_i^{(k)}\right)$. The code is written in Cython.

Let's now present the results. We used $\alpha = 0.0001$ as a constant that multiplies the regularization term. We also chose tol $= 0.001$ to provide the stopping criterion. We stop the algorithm when

$$\text{loss} > \text{best\_loss} - \text{tol}.$$

The results depended on the random number generator (choice of the samples) and varied throughout the experiments. One of the best results for each dataset are presented as follows.

Dataset	Training Acc.	Test Acc.	Non-zero coefficients
Breast Cancer	0.989	0.9649	25/30
NBA	0.5896	0.5746	7/12
Bankruptcy	0.695	0.724	11/17

Table 1: SGDClassifier Results

We have also tested how the value of loss function evolved with increasing number of time stamps.

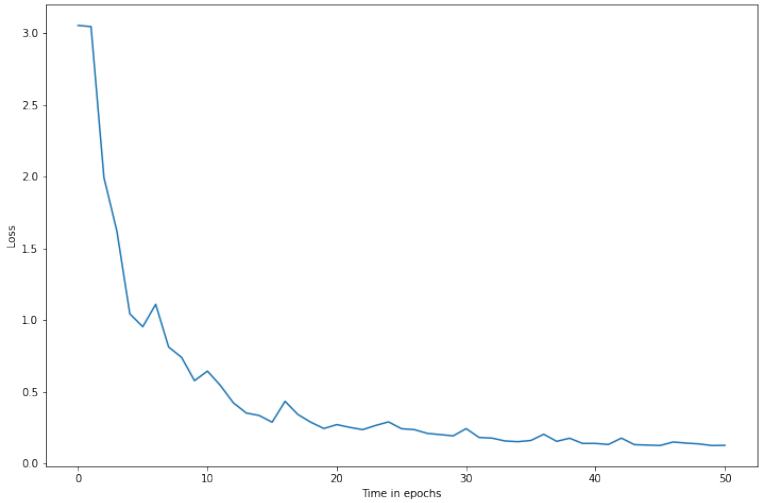

Figure 2: Loss in a specific time stamp for Breast Cancer dataset.

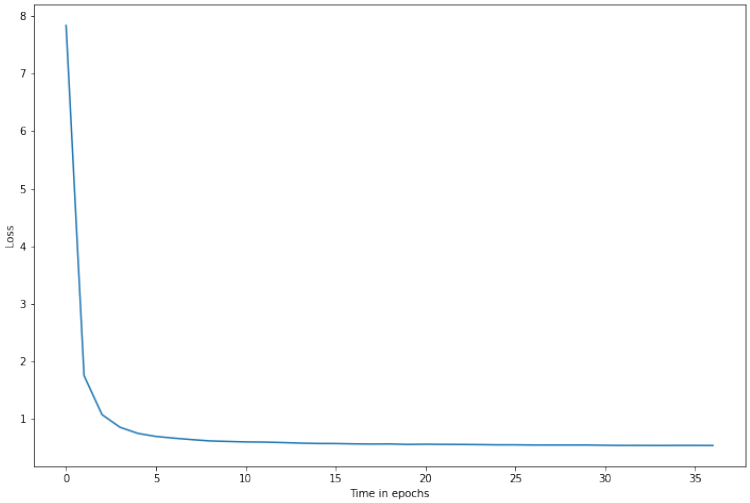

Figure 3: Loss in a specific time stamp for Bankrupcy dataset.

3.3 LogisticRegression

This class implements regularized logistic regression using the *liblinear* library (Sklearn documentation). L1 regularized logistic regression solves the same optimisation problem as for the other algorithms which was already mentioned. The used two different solvers in case of L1 regularisation:

- *liblinear* - uses a coordinate descent algorithm and relies on the excellent C++ LIBLINEAR library which is shipped with scikit-learn [6]. For L1 regularization *sklearn.svm.l1_min_c* allows to calculate the lower bound for C in order to get a non *null* model.
- *saga* - is a variant of *sag* solver (using Stochastic Average Gradient descent algorithm) that also supports the non-smooth L1 penalty. This solver is faster for large datasets.

In this model the constant that multiplies the regularisation term is inversely proportional to the constant α used in *SGDClassifier*. Based on the documentation, we assigned $C = N/\alpha$, where N is the sample size for both solvers. We used the same stopping criterion with tol = 0.001.

The results of the experiments are presented as follows.

Dataset	Training Acc.	Test Acc.	Non-zero coef
Breast Cancer	0.998	0.9386	30/30

Table 2: Liblinear Solver Results, tol = 0.001.

Dataset	Training Acc.	Test Acc.	Non-zero coef
Breast Cancer	0.9868	0.9824	30/30
NBA	0.7089	0.7425	12/12
Bankruptcy	0.69988	0.7325	17/17

Table 3: SAGA Solver Results, tol = 0.001.

After the experiments we noted that the model with *liblinear* solver is a bit over-fitted and does not perform well on new data. Therefore, we changed the stopping criterion parameter to 0.01. Afterwards, the results for the test set improved significantly.

Dataset	Training Acc.	Test Acc.	Non-zero coef
Breast Cancer	0.9868	0.9825	30/30
NBA	0.7118	0.7388	12/12
Bankruptcy	0.701625	0.737	17/17

Table 4: Liblinear Solver Results, tol = 0.01.

Although the norm of the weights vectors decreased, this model did not perform well as a feature selector.

3.4 Glmnet

Glmnet is a Python package based on the implementation in R. It is used to fit generalized linear and similar models via penalized maximum likelihood [8]. The regularization path is computed for the lasso (L1) penalty at a grid of values for the regularization parameter lambda. The glmnet algorithms use cyclical coordinate descent algorithm. It optimizes the objective function over each parameter with other fixed, and cycles until convergence.

In the experiments, we used 100 different values of lambda regularization parameter between $3.8 \cdot 10^{-5}$ and 0.38. We also provided the model with stopping criterion as in the previous algorithms with tol $= 0.001$.

Dataset	Training Acc.	Test Acc.	Non-zero coef
Breast Cancer	0.9824	0.9649	16/30
NBA	0.6866	0.7239	5/12
Bankruptcy	0.7015	0.7115	8/17

Table 5: Glmnet Solver Results For Best λ.

We have also plotted the change in the values of the coefficients with the change of the parameter λ. These kind of plots are also called *coefficient paths*.

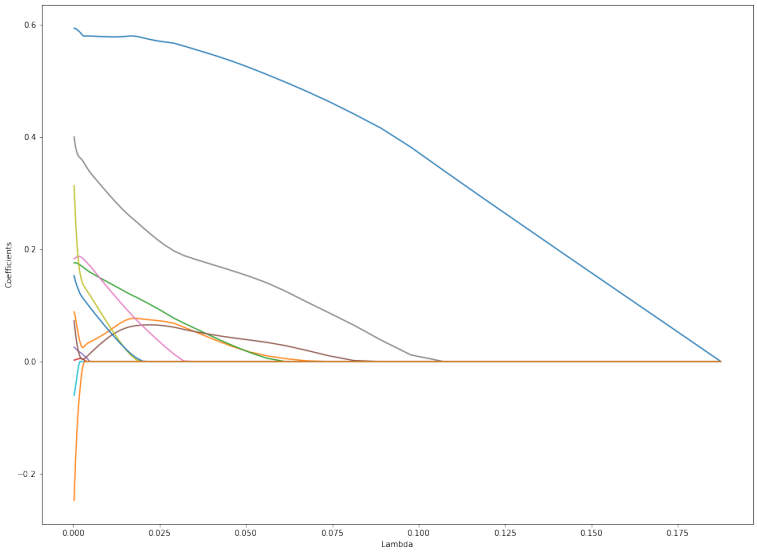

Figure 4: Dependance between the values of coefficients and λ for NBA dataset.

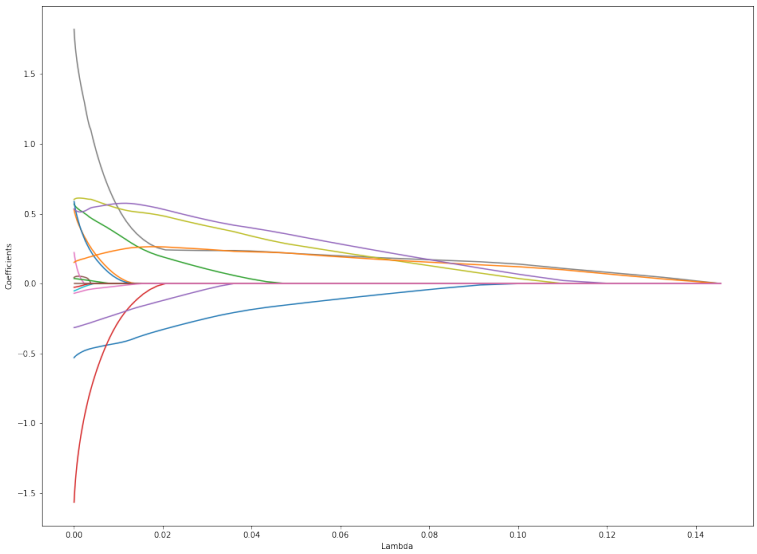

Figure 5: Dependance between the values of coefficients and λ for Bankruptcy dataset.

3.5 CVXPY

CVXPY is a Python-embedded modeling language for convex optimization problems [9]. It is mostly based on expressing the problem in a natural way that follow the math behind it. The package is not as restrictive as standard formulation required by solvers. The package relies on different open source solvers such as **ECOS**, **OSQP**, and **SCS**. Optimization algorithms are based on the book Convex Optimization by Boyd and Vandenberghe [10].

We used CVXPY to train a logistic regression classifier with L1 regularization. Similarly as in previous algorithms, we maximize the log-likelihood of the data minus a regularization term $\lambda\|w\|_1$. The optimization problem is formulated as follows.

```
beta = cp.Variable(n)
lambd = cp.Parameter(nonneg=True)
log_likelihood = cp.sum(
    cp.multiply(y_train, X_train @ beta) - cp.logistic(X_train @
    beta)
)
problem = cp.Problem(cp.Maximize(log_likelihood/m - lambd * cp.norm
    (beta, 1)))

def error(scores, labels):
  scores[scores > 0] = 1
  scores[scores <= 0] = 0
  return np.sum(np.abs(scores - labels)) / float(np.size(labels))
```

Listing 1: Formulation of Optimization Problem in CVXPY

We first define the variable based on the dimension of the data and parameter λ that is responsible for the strength of the regularization. Then, the log-likelihood is defined. The problem is implemented as maximizing scaled log-likelihood minus $\lambda\|w\|_1$. We define the error as mean of misclassified classes. Similarly as in *glmnet*, we use a range of values of λ between 0.01 and 1.0 (100 different values from log-space).

The best results for Breast Cancer dataset was obtained for $\lambda \approx 0.019$.

Dataset	Training Acc.	Test Acc.	Non-zero coef
Breast Cancer	0.9758	0.9912	10/30
NBA	0.6791	0.7201	10/12
Bankruptcy	0.714	0.7063	7/17

Table 6: CVXPY Results For Best λ

This method appeared to have the highest time of computation. We have also prepared plots of coefficients' paths for different values of λ and training and test errors for different values of λ.

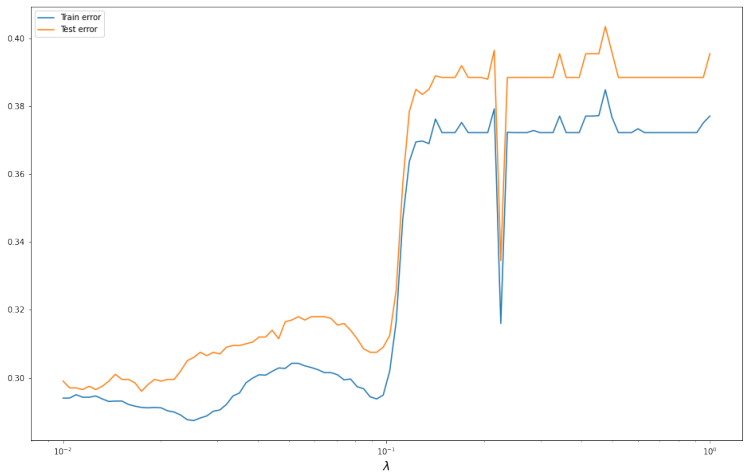

Figure 6: Test and training errors for Bankruptcy dataset.

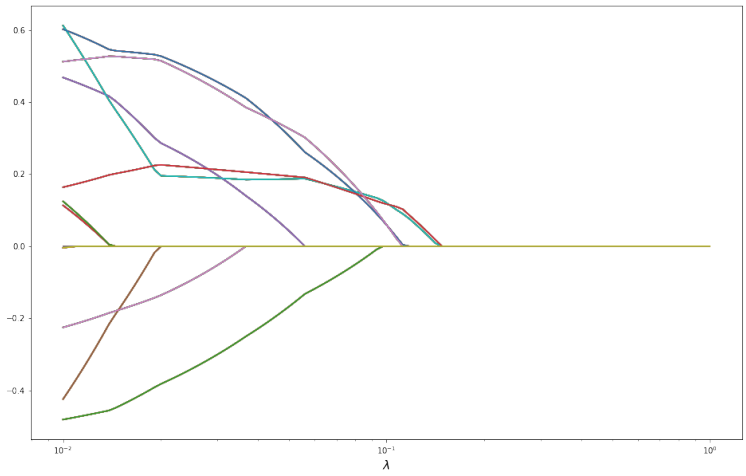

Figure 7: Coefficient paths for Bankruptcy dataset.

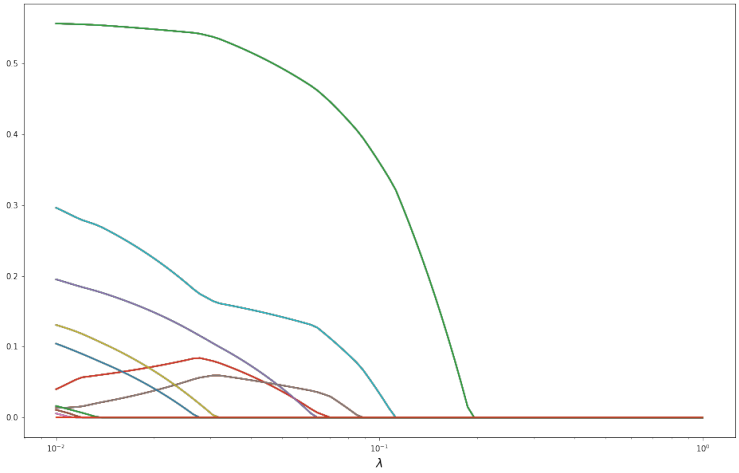

Figure 8: Coefficient paths for NBA dataset.

Note that the values of NBA dataset coefficients were all positive. It may be connected with the fact that each statistics of a player only increased the probability of being rewarded with MVP title.

4 Bibliography

[1] Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani. *Least angle regression*. The Annals of Statistics (2004)

[2] Su-In Lee, Honglak Lee, Pieter Abbeel and Andrew Y. Ng. *Efficient L1 Regularized Logistic Regression*. Computer Science Department Stanford University

[3] SGD Scikit-Learn documentation

[4] SGD Scikit-learn documentation in details

[5] Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Penalty, Yoshimasa Tsuruoka, Jun'ichi Tsujii, Sophia Ananiadou

[6] LogisticRegression Sklearn documentation

[7] Aaron Defazio, Francis Bach, Simon Lacoste-Julien: SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives

[8] https://glmnet.stanford.edu/articles/glmnet.html#logistic-regression-family-binomial-

[9] CVXPY Documentation

[10] Boyd Stephen, Vandenberghe Lieven: Convex Optimization