CSCI-SHU 360 Machine Learning
Solution to homework 3

Josiah Li yl111912@nyu.edu

April 7, 2025

1 Programming Problem: Logistic Regression

1.1

By extended representation, we have $X \leftarrow [X, 1] \in \mathbb{R}^{n\times(d+1)}$, and $W \leftarrow [W, b] \in \mathbb{R}^{(d+1)\times c}$. For X_i, we have $z_i = X_i W$. And we have $Pr(y = k|x = X_i; W) = \frac{\exp(z_k)}{\sum_{j=1}^{c}\exp(z_j)} = P_{i,k}$

For the loss function $F(x)$, we have:

$$\begin{aligned}
F(x) &= \frac{1}{n}\sum_{i=1}^{n} -\log[Pr(y = y_i|x = X_i; W)] + \frac{\eta}{2}\|W\|_F^2 \\
&= \frac{1}{n}(\sum_{i=1}^{n} -(z_{y_i}) + \sum_{i=1}^{n}\log\sum_{j=1}^{c}\exp z_j) + \frac{\eta}{2}\|W\|_F^2 \\
&= \frac{1}{n}(-\sum_{i=1}^{n} X_i W_{:,y_i} + \sum_{i=1}^{n}\log\sum_{j=1}^{c}\exp X_i W_{:,j}) + \frac{\eta}{2}\|W\|_F^2 \\
&= \frac{1}{n}(-\sum_{i=1}^{n} X_i W_{:,y_i} + \sum_{i=1}^{n}\log\sum_{j=1}^{c}\exp X_i W_{:,j}) + \frac{\eta}{2}\sum_{j=1}^{c}\|W_{:,j}\|_F^2
\end{aligned}$$

Thus we take the partial derivative:

$$\begin{aligned}
\frac{\partial F(X)}{\partial W_j} &= \frac{1}{n}(-\sum_{i=1}^{n}\mathbf{1}\{y_i = j\}X_i^T + \sum_{i=1}^{n}\frac{\exp(X_i W_j)X_i^T}{\sum_{k=1}^{c}\exp(X_i W_k)}) + \eta W_j \\
&= \frac{1}{n}\sum_{i=1}^{n}(P_{i,j} - \mathbf{1}\{y_i = j\})X_i^T + \eta W_j
\end{aligned}$$

If we represent the derivative in numerator layout, we have $\frac{\partial F(W)}{\partial W} = \left[\frac{\partial F(W)}{W_1}, \cdots, \frac{\partial F(W)}{W_c}\right]$ Let $\Pi_{i,j} = \mathbf{1}\{y_i = j\}$, we have:

$$\frac{\partial F(W)}{\partial W} = \frac{1}{n}X^T(P - \Pi) + \eta W$$

Thus, after having the gradient of the function, we can have the gradient descent rule for W which is

$$W \leftarrow W - \lambda(\frac{1}{n}X^T(P - \Pi) + \eta W).$$

Where λ is the learning rate.

1.2

As we know, we have:

$$\frac{\exp(a)}{\exp(b)} = \frac{\exp(a-c)}{\exp(b-c)}.$$

So we have:

$$\frac{\exp(z_i)}{\sum_{j=1}^{c}\exp(z_j)} = \frac{\exp(z_i - \max_j z_j)}{\sum_{j=1}^{c}\exp(z_j - \max_j z_j)}.$$

Which we can see that this modification is the same as origin softmax function.
Moreover, the original form may lead to an extreme exponential value and lead to a numerical error.
We have the following output:

Learning Rate	Converged Iteration	Training Precision	Test Precision	Final Loss
0.05	111	0.9829	0.9667	0.2684
0.005	335	0.9762	0.9644	0.1859
0.01	219	0.9770	0.9644	0.1795

Table 1: Comparison of model performance under different learning rates

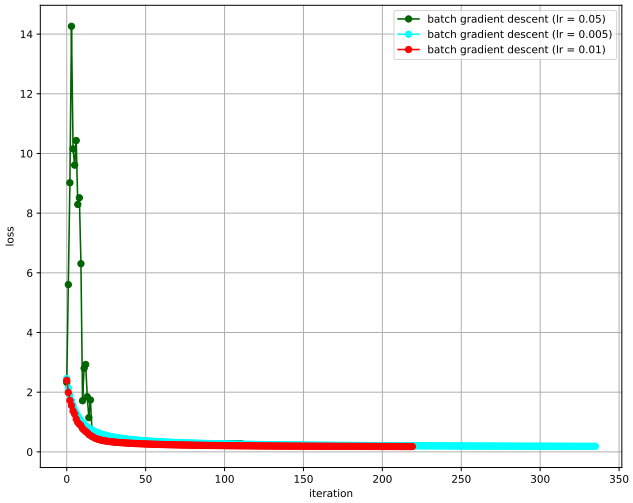

Figure 1: Convergence curves under different learning rate.

1.3

As we can see from the graph, for small learning rate, the model is converging in a more stable way but takes more iteration to converge, while the large learning rate will make the convergence less stable, but faster to converge.

1.4

We have the following output:

Learning Rate	Batch Size	Final Epoch	Training Precision	Test Precision	Final Loss
0.01	10	21	0.996	0.971	0.4623
0.01	50	21	0.985	0.971	0.2724
0.01	100	31	0.984	0.967	0.2590
0.005	10	21	0.992	0.973	0.3296
0.005	50	21	0.982	0.964	0.2491
0.005	100	26	0.972	0.960	0.2487
0.001	10	33	0.983	0.976	0.2538
0.001	50	58	0.975	0.956	0.2429
0.001	100	106	0.973	0.962	0.2447

Table 2: Performance comparison across different learning rates and batch sizes

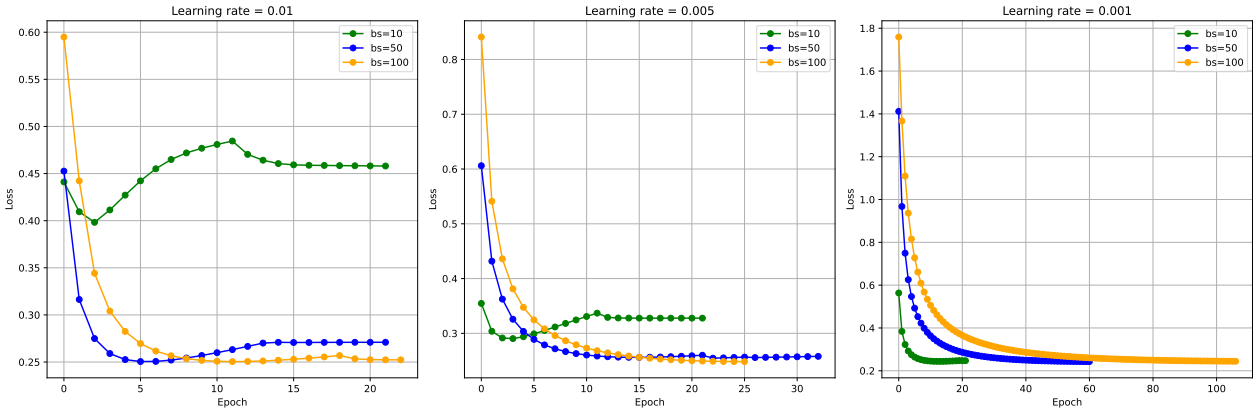

Figure 2: Convergence curves under different batch size and learning rate.

With the data and figure above, we can see that for the learning rate is small(0.001), the convergence are smooth for all three batch sizes, however, the test precision is generally lower than the other two learning rate settings. And the model takes more time to converge. For the large learning rate(0.01), the model tend to be harder to converge. But under this setting, the $bs = 50$ and $bs = 100$ are converging in a much faster speed comparing to the other two learning rate.

1.5

From the previous figure we can see that three batch sizes have different convergence speed with the same initial learning rate. By the instruction from the question, we use $\frac{10^{-2} \times bs}{100}$ as the initial learning rate for each batch size bs. The result is:

Batch Size	Final Epoch	Training Precision	Test Precision	Final Loss
10	22	0.9814	0.9689	0.2476
50	22	0.9822	0.9733	0.2495
100	21	0.9807	0.9711	0.2527

Table 3: Model performance under different batch sizes with adaptive learning rate

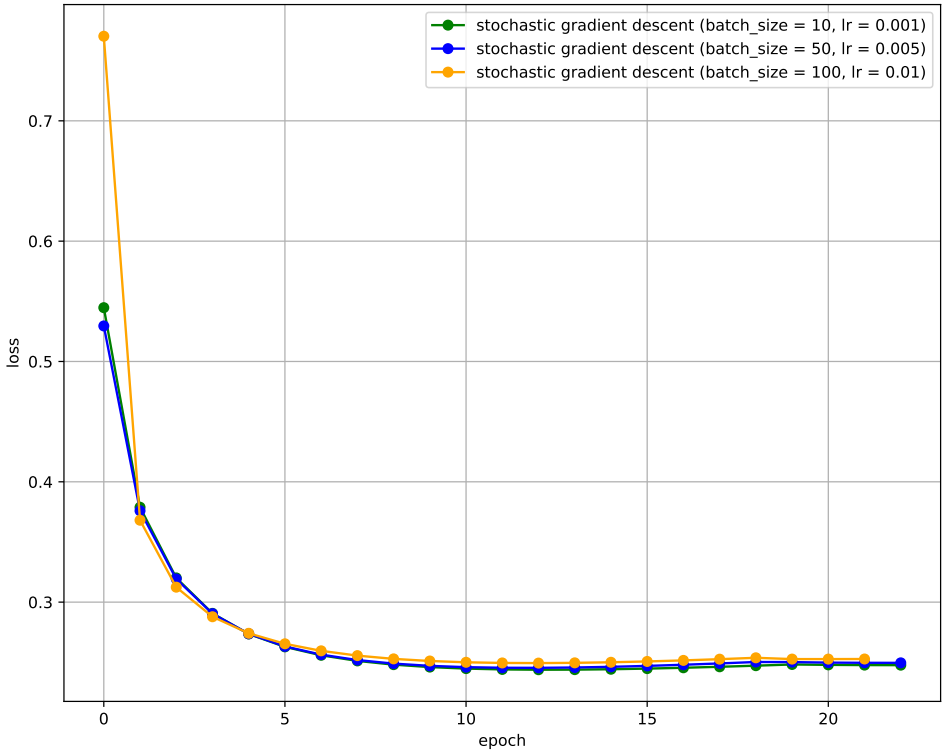

Figure 3: Convergence curves by applying different initial learning rate.

We can see from the figure that with initial learning rates adjusted according to the batch size, the convergence for all three models became much similar.

Explanation: when we are doing the SGD, we have $\Delta W_{epoch} = \frac{N}{B} \times \nabla F_{batch}$. N is the total number of samples and B is the batch size. We can see that as ∇_{batch} is approximately the general gradient of the dataset. Thus, for every epoch, the smaller batch size leads to larger gradient change per epoch.

2

2.1

After performing the coordinate descent, we can have the following curve.

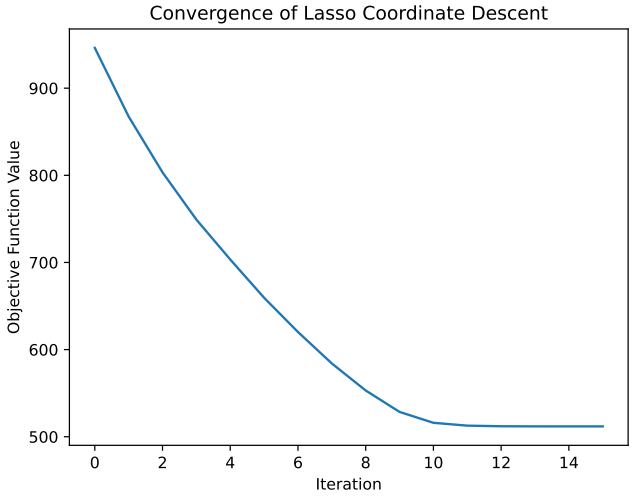

Figure 4: The convergence curve of the Lasso Coordinate Descent.

The indices of non-zero weight entries are: [0, 1, 2, 3, 4, 11, 12, 25, 34, 36, 52, 63, 71].

2.2

We can get the metrics below:

Metric	**Value**
RMSE	0.8265
Sparsity of w	13
Precision of w	0.3846
Recall of w	1.0000

Table 4: Evaluation Metrics of the Lasso Model

2.3

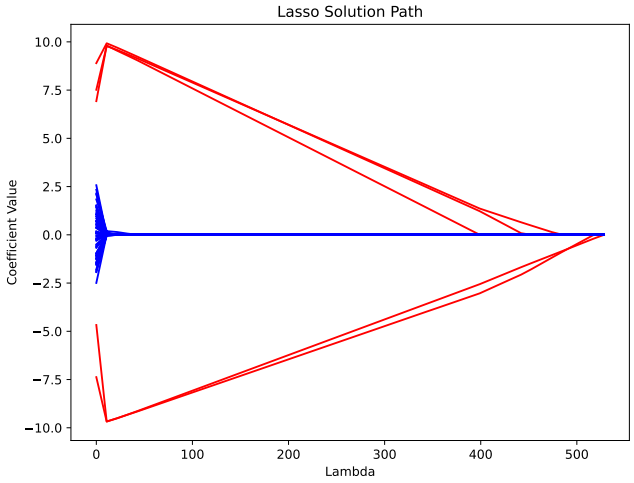

Figure 5: Lasso solution path for the entries.

From the figure above, we can see that the features are generally saperated into two classes. One class converges to zero in a quick speed while another class will first have a peak before descending. The first class tend to be the zero entries in the real parameters while the other is the non-zero entries in θ. From

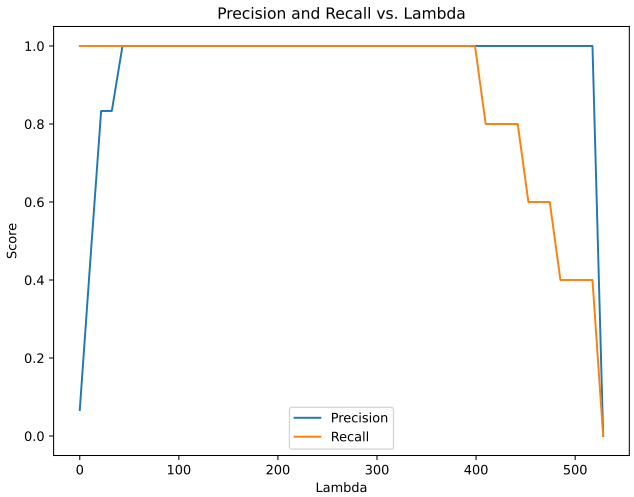

Figure 6: Precision and recall for different λ.

the figure above, we can see that the when we have a large λ the model tend to reduce the scale of θ by setting most of the entries to zero. And when we have a smaller λ, the model will have less punishment on the scale of the parameters which lead to a high recall rate. Thus, the appropriate λ will pick the valuable entries and abandon the rest.

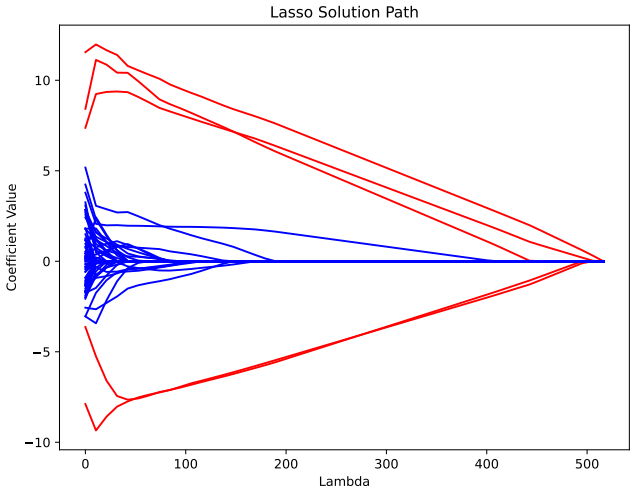

Figure 7: Lasso solution path for the entries when $\sigma = 10$.

We can see that conparing to the previous figure, with larger σ the convergence of parameters are much more unstable. And as the blue features (the irrelevant features) might have non-zero weight due to the large noise. Under this circumstances, tuning λ become much more important.

2.4

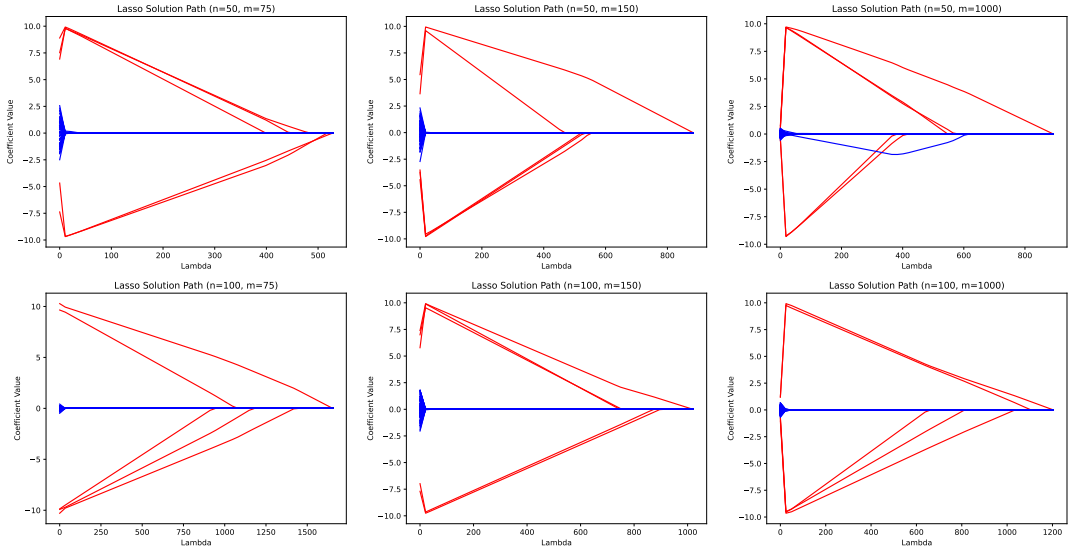

Figure 8: The Lasso solution path under different n and m.

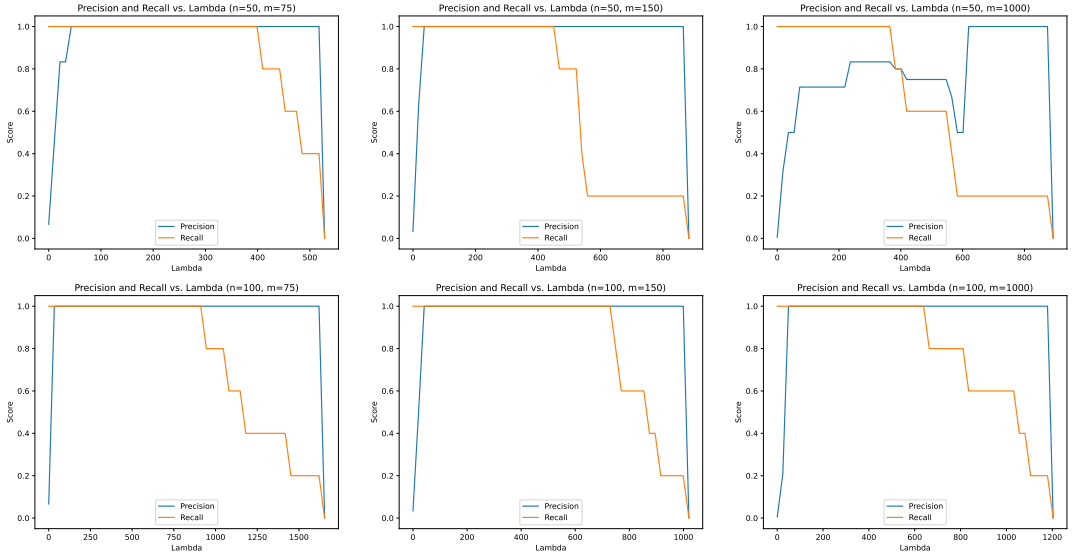

Figure 9: The precision and recall under different n and m.

According to the raw data, we have te table below. To be noted, for $n = 50, m = 1000$ we are not able to find λ that has both precision and recall to be 1. Thus, if we have to find a optimal λ, by searching for the max sum of precision and recall, it should be $[236, 365]$(round to integer).

Table 5: The optimal lambda under different n and m(round to integer).

$n\backslash m$	75	150	1000
50	[43, 399]	[36, 450]	[236, 365]*
100	[34, 911]	[42, 729]	[49, 639]

2.5

Implementation:
First, the naive implementation was designed for dense matrices and small datasets. And for large and sparse data in Sub-Problem 5, we can use Compressed Sparse Column to optimize the calculation. As csc type is more effcient for column slicing and get non-zero entries in the matrices.
In the naive implementation, we are calculating the residuals for the jth coordinate by doing an expensive matrix multiplication, which is $O(n \times m)$ for every coordinate, taking consideration of the total iteration loop T and number of coordinates, the total time complexity will become $O(Tnm^2)$. In contrast, if we calculate the residuals outside the inner loop and update the residuals dynamically update the residual matrix. The time complexity for every coordinate will reduce to $O(n)$, and the total time complexity will reduce to $O(Tnd)$. Moreover, as we did some small tricks by using csc datatype, we are able to only update the non-zero entries, which reduce the cost in the sparse case.

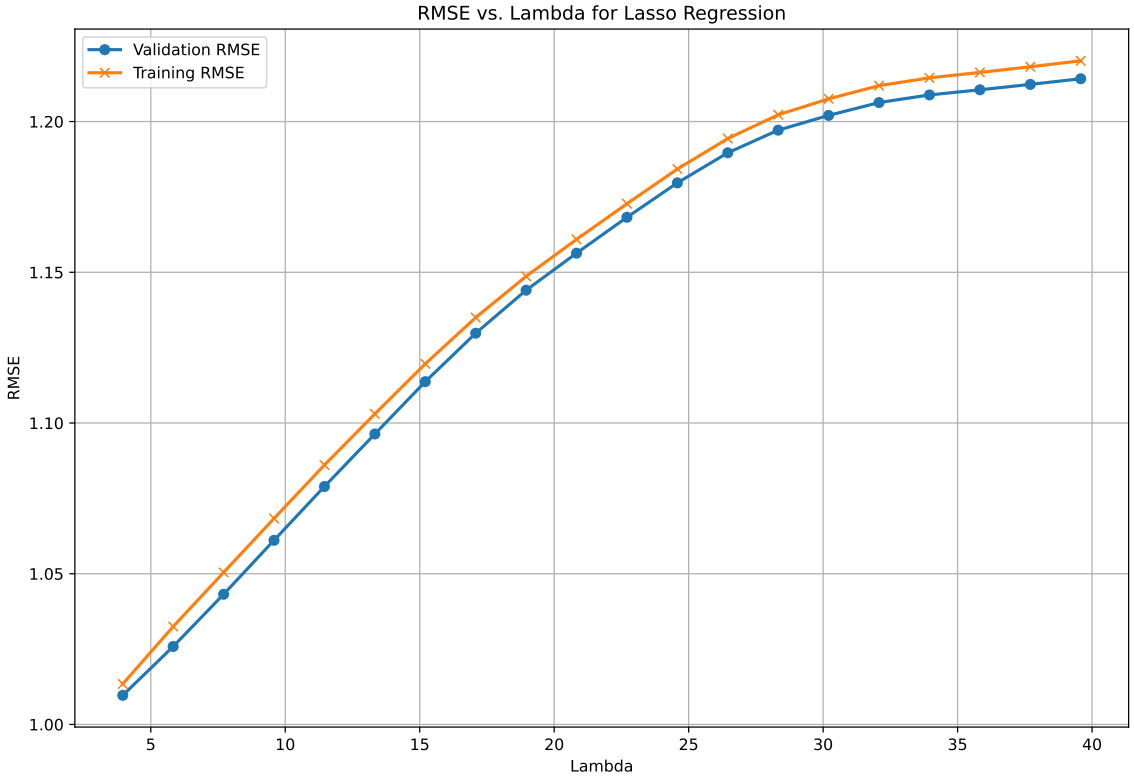

Figure 10: training and validation rmse v.s. λ values.

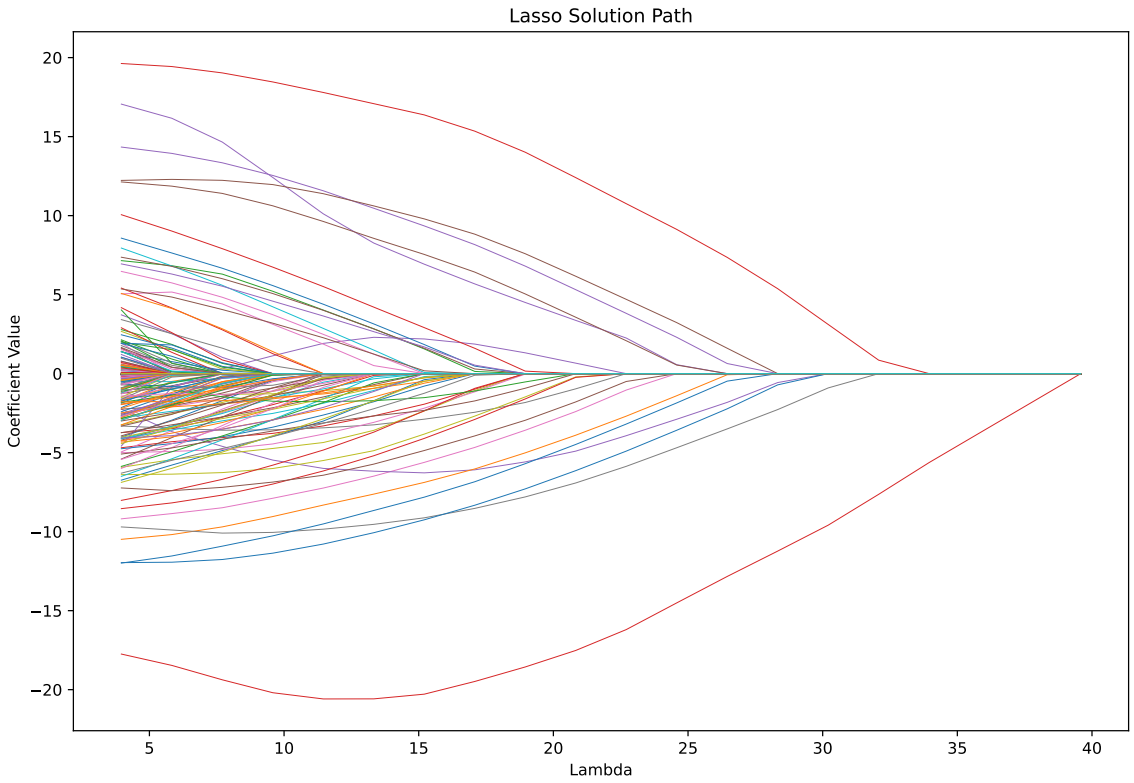

Figure 11: the lasso solution path.

Best lambda: 3.95782540136483
Validation rmse: 1.0096512302660232

feature	**weight**
great	19.626910602812263
not	-17.746498530426216
best	17.05719025532021
amazing	14.339641440305963
love	12.228556228270259
delicious	12.132458330783074
rude	-11.989863473044203
the worst	-11.959659903513517
horrible	-10.486167300825196
awesome	10.058457548734463

Table 6: top 10 lasso weights.