CSCI-SHU 360 Homework3

Ninghao Lu nl2752

2024 Fall

Exercise 1 Logistic Regression

1.1 Solution:

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

For simplicity, let $F_1(W) = X_iW_{y_i}$, $F_2(W) = log\sum_{j=1}^{c} exp(X_iW_j)$, $F_3(W) = \frac{\eta}{2}\sum_{j=1}^{c}||W_j||_F^2$, then we have

$$
\frac{\partial F_1(W)}{\partial W_j} = -\frac{1}{n}\cdot\begin{cases} x, & \text{if } j = y_i \\ 0, & \text{if } j \neq y_i \end{cases}
$$

$$
\frac{\partial F_2(W)}{\partial W_j} = \sum_{j=1}^{c}\frac{1}{exp(X_iW_j)}\cdot exp(X_iW_j)X_i^T = Pr(y=j|X_i;W)X_i^T
$$

$$
\frac{\partial F_3(W)}{\partial W_j} = \eta W_j
$$

Therefore, $\frac{\partial F(W)}{\partial W_j}$ becomes,

$$
\begin{aligned}
\frac{\partial F(W)}{\partial W_j} &= -\frac{1}{n}\sum_{i=1}^{n}\mathbb{1}\{y_i=j\}X_i^T + \frac{1}{n}\sum_{i=1}^{n}Pr(y=j|X_i;W)X_i^T + \eta W_j \\
&= \frac{1}{n}\sum_{i=1}^{n}(Pr(y=j|X_i;W) - \mathbb{1}\{y_i=j\})X_i^T + \eta W_j \quad \text{Summing up everything, we can write it in matrix form} \\
&= \frac{1}{n}X^T\begin{pmatrix} Pr(y=j|X_i;W) - \mathbb{1}\{y_1=j\} \\ Pr(y=j|X_2;W) - \mathbb{1}\{y_2=j\} \\ \cdots \\ Pr(y=j|X_n;W) - \mathbb{1}\{y_n=j\} \end{pmatrix} + \eta W_j
\end{aligned}
$$

Define matrix $P \in R^{n\times c}$ where $P_{ij} = Pr(y=j|X_i;W)$, define matrix $Y \in R^{n\times c}$ where $Y_{ij} = \mathbb{1}\{y_i=j\}$.

$$
\frac{\partial F(W)}{\partial W} = \left[\frac{\partial F(W)}{\partial W_1}\cdots\frac{\partial F(W)}{\partial W_c}\right] = \frac{1}{n}X^T(P-Y) + \eta W
$$

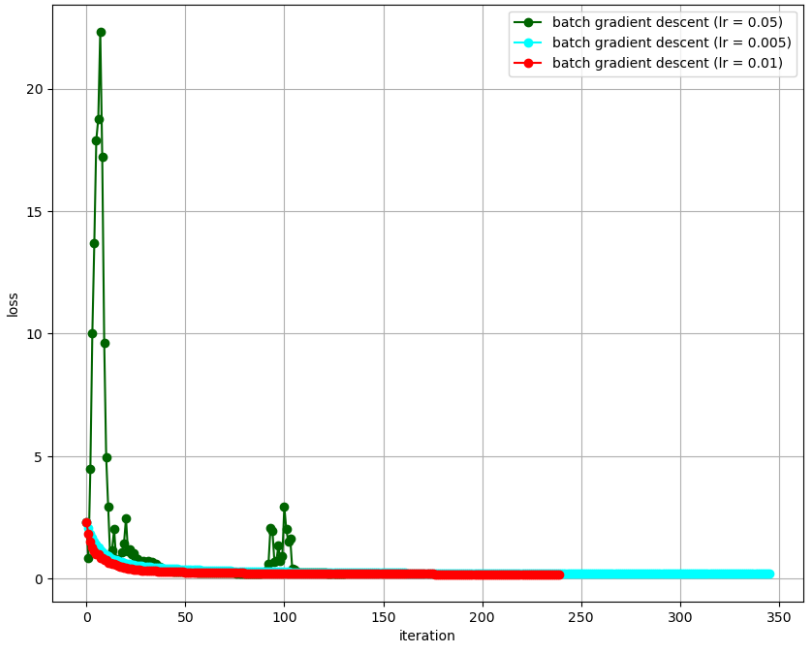
Therefore the gradient descent rule for W is

$$W^{(t+1)} \leftarrow W^{(t)} - \alpha \frac{1}{n} X^T (P - Y) - \alpha \eta W$$

where α is the learning rate.

1.2 Solution:

After implementation, we get the following result:



Why such a modification can avoid numerical problems?

By subtracting the maximum logit, all the other logits are non-positive and this prevents e^{z_k} becomes too large when applying the softmax function so that it will not exceed the range of floating point numbers.

Why the overall result remains unchanged after the modification?

Since it is only a shift by constant, the probabilities assigned by softmax is still proportional to the original logits, i.e.

$$\begin{aligned}
\frac{e^{z^k - max_j z_j}}{\sum_j e^{z_j - max_j z_j}} &= \frac{e^{z^k} / e^{max_j z_j}}{\sum_j (e^{z_j} / e^{max_j z_j})} \\
&= \frac{e^{z^k} / e^{max_j z_j}}{1 / e^{max_j z_j} \sum_j (e^{z_j})} \\
&= \frac{e^{z^k}}{\sum_j (e^{z_j})}
\end{aligned}$$

Therefore, the modification doesn't change the overall result.

1.3 Solution:

For large learning rates: The advantage of larger learning rate is that it converges faster. (For learning rate 0.01, it runs 239 epochs and for learning rate 0.005, it runs 345 epochs). The disadvantage of large learning rate is that the training process is not very stable as we can see there are spikes in the curve when

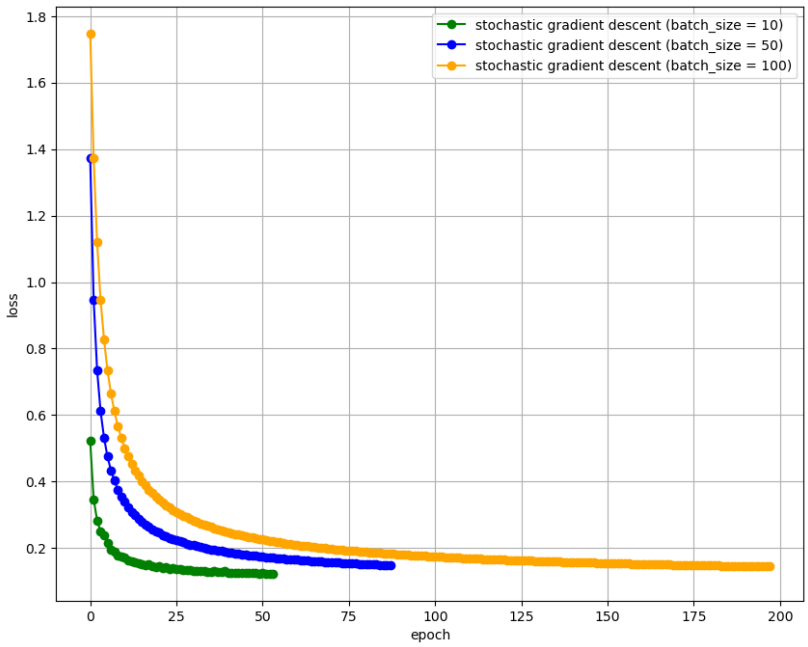
learning rate is 0.05. But the training processes of lr = 0.005 and lr = 0.01 are smoother. Apart from what we can see from the graph, large learning rate may miss the global minimum or divergence.
For small learning rates: The advantage of smaller learning rate is that it has a smoother and more stable training process. The disadvantage is that it requires more epochs to train the model which results in slower speed in convergence.

1.4 Solution:

After the implementation, we get final value of F(W) and final training and test accuracy in the following table:

Batch Size	Training accuracy	Test accuracy
10	0.988864	0.971111
50	0.978471	0.966667
100	0.979955	0.971111

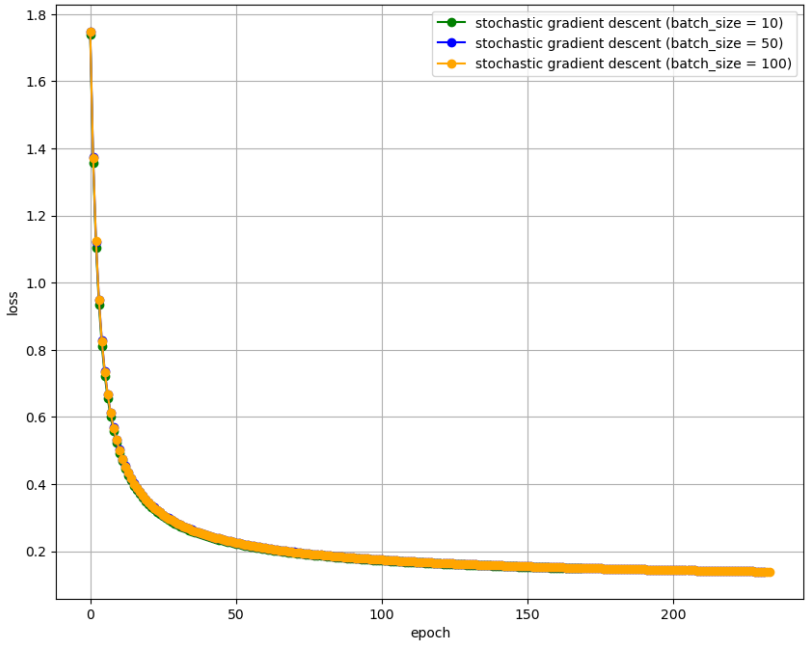
We also get the following plot containing the three convergence curves.



1.5 Solution:

When the same initial learning rate is used, the convergence curves don't show the same convergence speed according to the graph above.
Then, we will scale the learning rate linearly with the batch size. We initialize the learning rate = 0.001 for batch size 10. Therefore, the learning rates for batch size 50 and 100 are 0.005 and 0.01 respectively. Then we get the following table and the plot.

Batch Size	Learning Rate	Training Precision	Test Precision
10	0.0001	0.976244	0.964444
50	0.0005	0.982925	0.973333
100	0.0010	0.983667	0.973333



Compared to old convergence curves, we can see that different convergence curves have approximately the same convergence speed.

Mathematical explanations to this phenomenon

When using mini-batch SGD, the update to W is based on:

$$\nabla F(W) = \frac{1}{b} \sum_{i \in \text{mini-batch}} \nabla F_i(W)$$

where b is the mini-batch's size. The update rule for the weights in mini-batch SGD is:

$$W := W - \alpha \nabla F(W)$$

where α is the learning rate, it controls the step size of each update. From the two formula above we can see a trade-off between b and α. If we decrease the batch size, the estimate of $\nabla F(W)$ becomes noiser because few data points are involved in the calculation and this could prevent model from converging. To balance this, if we scale the learning rate linearly with the batch size, we will have

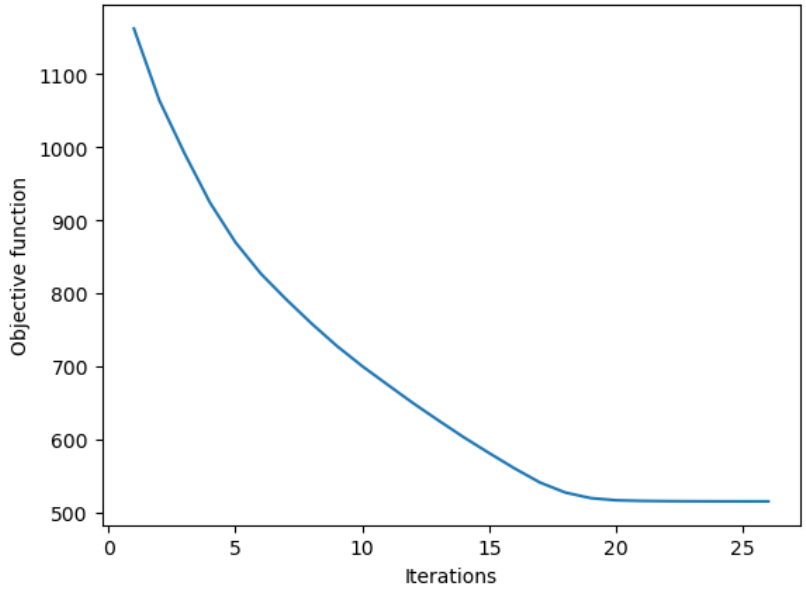
$$\alpha_{\text{new}} = \alpha_{\text{old}} \times \frac{b_{\text{new}}}{b_{\text{old}}}$$

This ensures that as the batch size increases, the learning rate also increases to have faster convergence. While when the batch size decreases, the learning rate will also get smaller and this makes the convergence speed slower.

Exercise 2 Lasso

2.1 Solution

After implementing coordinate descent, we get the plot of $F(\theta, \theta_0)$ v.s. coordinate descent iterations as follows and it is consistent with the expectation that it is non-increasing:



Moreover, the indices of non-zero weight entries are [0, 1, 2, 3, 4, 12, 16, 25, 34, 36, 52, 63, 71].

2.2 Solution

$$\text{RMSE} = 0.8593$$

$$\text{Sparsity} = 13$$

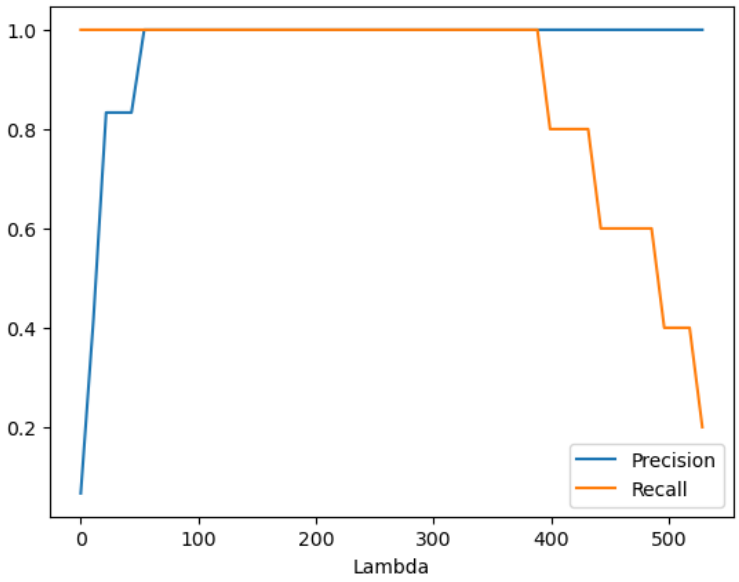
$$\text{Precision} = 0.3846$$

$$\text{Recall} = 1.0$$

2.3 Solution

2.3.1 The precision v.s. λ and recall v.s. λ curves

The precision v.s. λ and recall v.s. λ curves are shown below.



Some discoveries from the plot:

1. When λ is very small

Since λ is quite small, there is little penalty on the weights and this allows more features to join in the process of fitting data. Therefore, the model will make more weights non-zero in order to cover more non-zero indices in θ^*. As a result, the recall is high because the model includes those important features in the true model but the precision is low because the model also includes many irrelavent features.

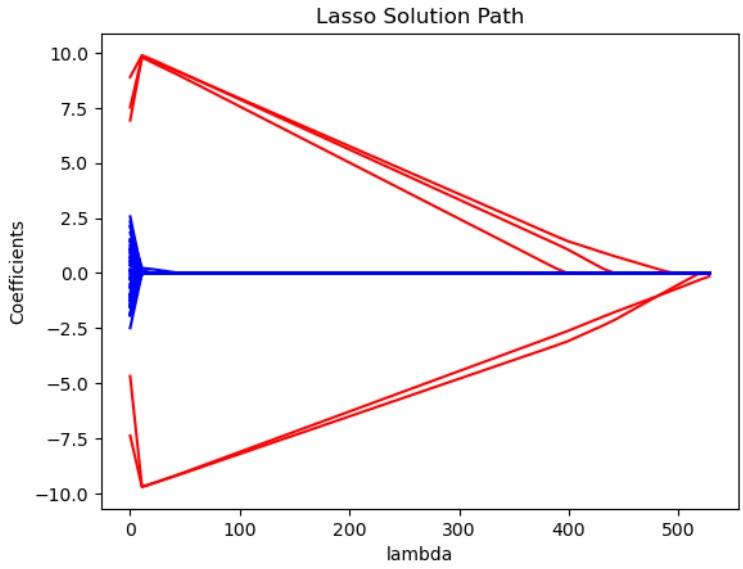
2. When λ is just right

In this range, the Lasso model is able to correctly select the non-zero features and discard irrelavent features and makes both precision and recall be 100%.

3. When λ is too large

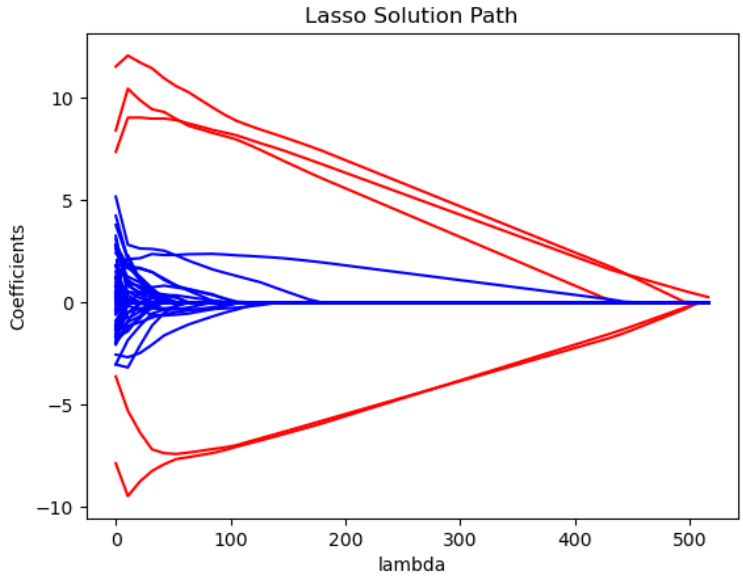
The penalty on the weights becomes stronger and pushes many weights towards zero. Only really essential features are kept, which results in high precision; On the other hand, the strong penalty also makes some of the non-zero indices not be captured and this will result in low recall.

2.3.2 Lasso solution path



As shown in the plot, we observed how the weights change as λ increases. The weights corresponding to non-zero indices are plotted in red, while the weights for zero entries are shown in blue. We can see the blue curves are quickly going down to 0, indicating that the model is removing irrelavent features while the red curves decreases much slower and gradually they converge to 0 as well.

2.3.3 Lasso solution path with $\sigma = 10.0$

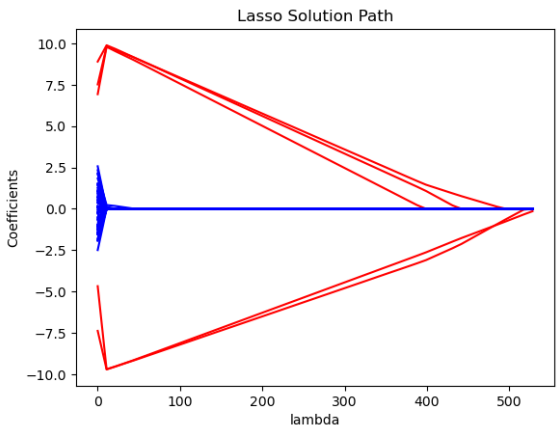


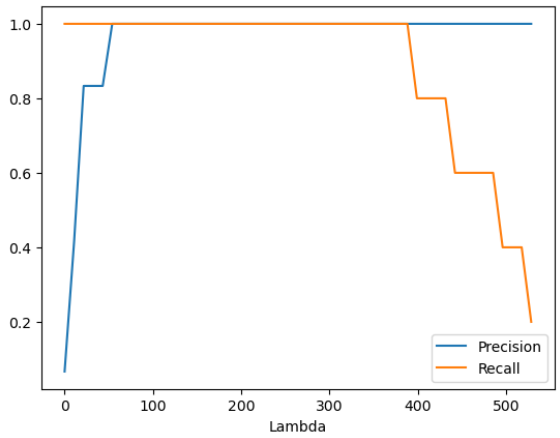
1. With higher noise ($\sigma = 10.0$), we can see that the blue curves converge much slower and noiser than the previous one. This suggests that the model is having a harder process in differentiating the relavent and irrelavent features.
2. We can see that there's a blue curve remains non-zero than a red curve, meaning that an irrelavent is kept while a relavent feature is removed. This will affect the calculation of precision and recall because the model may not reach a good score (have a high recall and a high precision at the same time) on both metrics. Therefore, the model will have a poorer performance in distinguishing relavent and irrelavent features.

2.4 Solution

After implementation, we get the results displayed below.

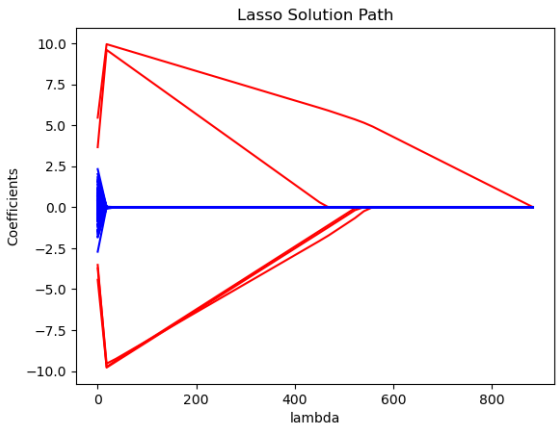
(n = 50, m = 75)

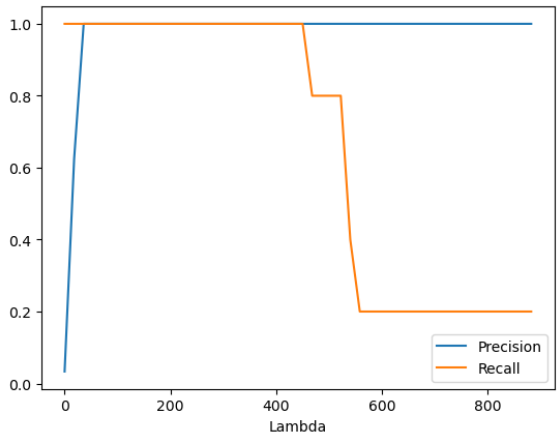




From the plots we can see that the values of λ that result in both good precision and recall are $\lambda \in [53.90376553492914, 388.10711185148983]$

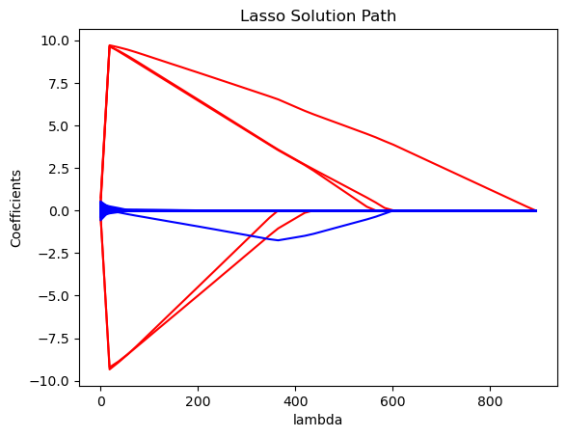
(n = 50, m = 150)

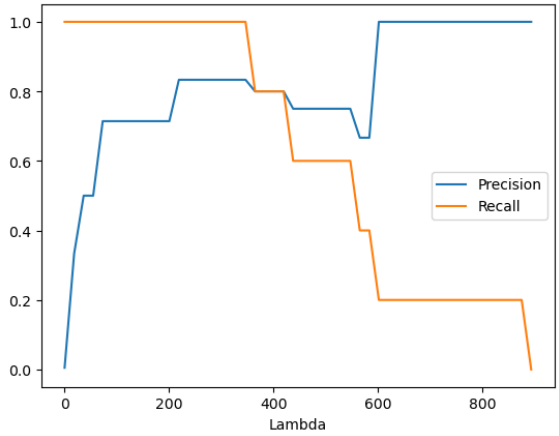




From the plots we can see that the values of λ that result in both good precision and recall are $\lambda \in [36.03149416132464, 450.39367701655794]$

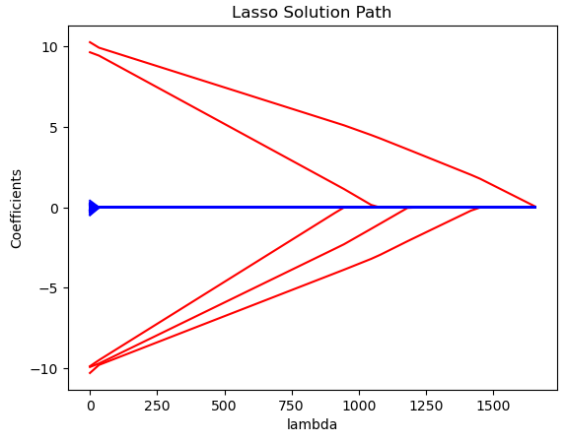
(n = 50, m = 1000)

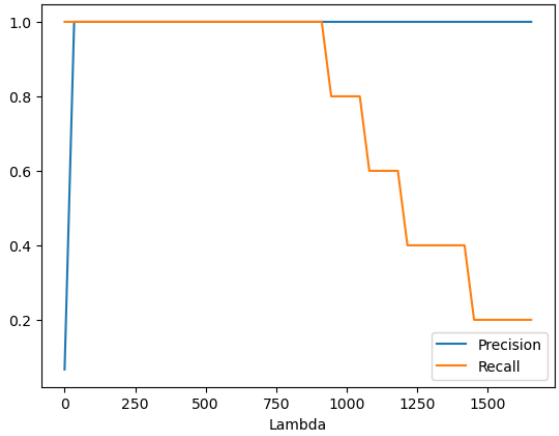




From the plots we can see that the values of λ that result in both good precision and recall are $\lambda \in [364.45738253422655, 419.1259899143605]$

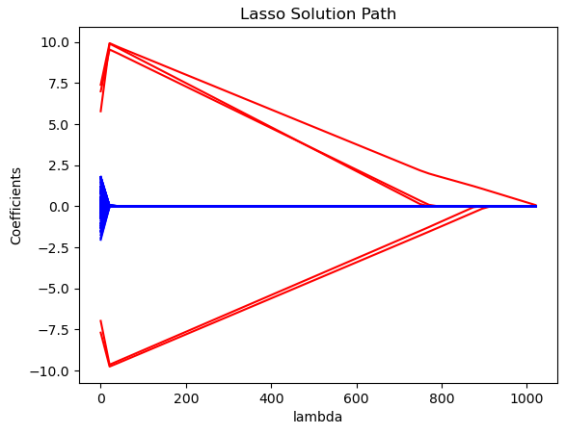
(n = 100, m = 75)

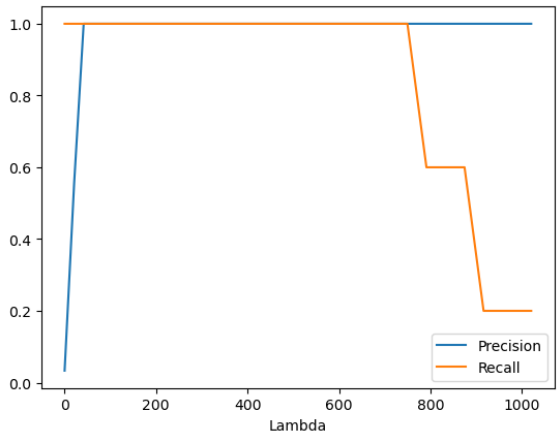




From the plots we can see that the values of λ that result in both good precision and recall are $\lambda \in [33.755662095952886, 911.4028765907279]$

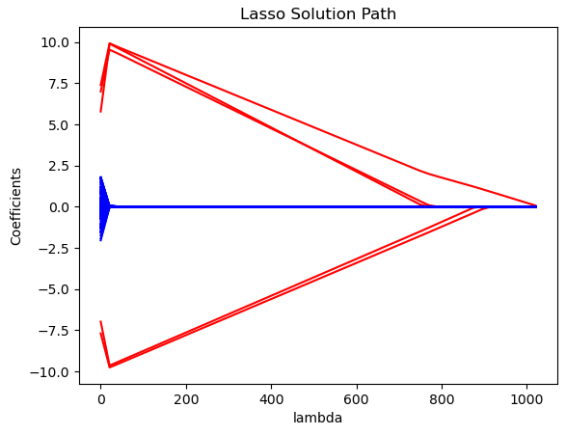
(n = 100, m = 150)

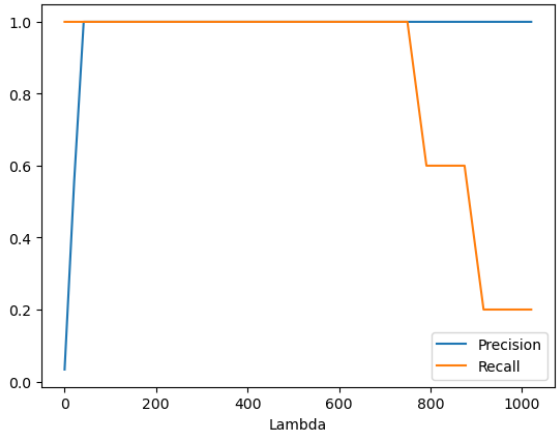




From the plots we can see that the values of λ that result in both good precision and recall are $\lambda \in [41.640431997869165, 749.5277759616449]$

(n = 100, m = 1000)





From the plots we can see that the values of λ that result in both good precision and recall are $\lambda \in [73.70493900078691, 663.3444510070822]$

2.5 Solution

Modification to the original implementation

(1) In the updated implementation, y (target) are transformed into scipy.sparse matrices and X_train has already been transformed during the data processing process so that they only store the non-zero elements along with their row and column indices, which can reduce memory usage a lot.
(2) In updated lasso, a new way to calculate c_j is introduced, and now we use ρ_j for soft thresholding implementation. originally, c_j is updated as

$$c_j = X_j^T \cdot (y - (X \cdot w - X_j w_j))$$

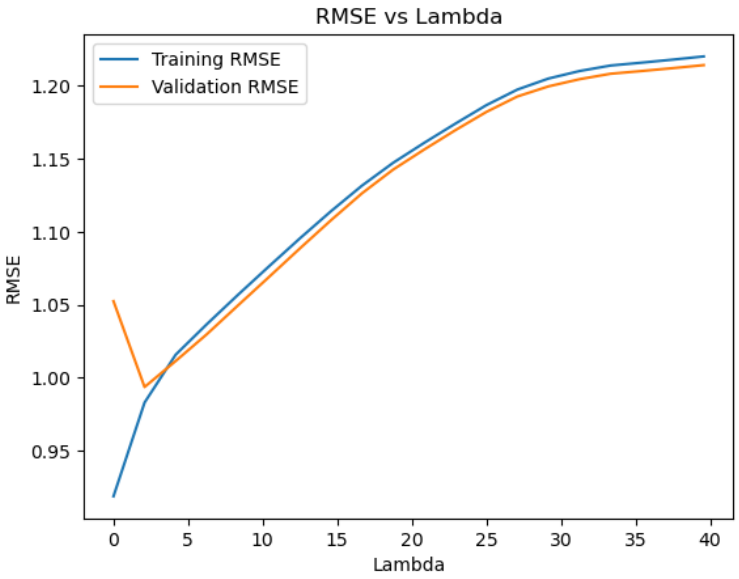
and this can be transformed into ρ_j to avoid recalculating X · w as we can break c_j down as

$$\rho_j = X_j^T \cdot r + z_j \cdot w_j$$

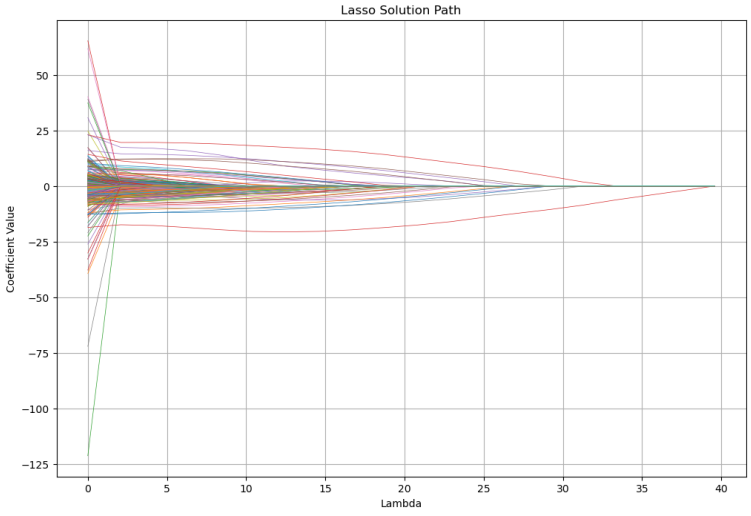
where $z_j = X_j^T X_j$. Therefore, we can calculate c_j in a more efficient way.
(3) In the original implementation, during coordinate descent, the residual was calculated in every iteration,

which is quite time-consuming. However, in our updated version, I only calculated residual at the very beginning outside the inner loop. Moreover, when updating w[j], I avoid calculating entire X.dot(w) by updating only those entries in the residual affected by changes in w[j] (see function $updated_lasso$ in the code file).

Plots on training RMSE & validation RMSE v.s. λ values



Lasso solution path



Report

According to my result, the best λ value is 2.0830660007183326 and the corresponding testing RMSE is 1.0458263808250714.
The top-10 features (words in comments) with the largest magnitude in the lasso solution w when using the best λ value are The reason why they are meaningful because the selected features are all adjectives that can

Feature	**Weight**
great	19.598197620019985
not	-17.418697434531826
best	17.407623494338292
amazing	14.494656180063801
rude	-12.446925025272716
love	12.151673818860884
the worst	-12.091188158926947
delicious	11.982090500844862
awesome	11.296925554692555
horrible	-10.719387727114022

Table 1: Top 10 Lasso-selected features and their weights

reflect the attitude of the comment. Moreover, their weights can also reflect to what extent, the comment has certain feelings.