Optimization

Project: Elastic Net Regularization

Authors: | *Student Numbers:*
Jelle de Rooij 1250750
Robbert Manders 1263872

December 4, 2020

1 BOAR report

We are researching different kinds of optimization algorithms for the Elastic Net regularization problem (H. Zou and T. Hastie (2005)). The Elastic Net regularization is defined as follows:

$$\hat{\beta}_{EN} = \arg\min_x \frac{1}{2n}||y - X\beta||_2^2 + \frac{\lambda_2}{2}||\beta||_2^2 + \lambda_1||\beta||_1$$

As you can see, this is a linear combination of the Lasso and Ridge penalty optimization problems. It has the advantage of Lasso that introduces sparsity in the final solution, while the Ridge penalty gives a more stable solution.

(a)

To solve all this problem we use MATLAB in combination with YALMIP and the GUROBI solver. For consistency between models, we set the random seed to our group number, 26. We used a grid-search to find the values of (λ_1, λ_2). We searched in: $[0, 0.001, 0.01, 0.1, 1, 10]$ for both λ's. To evaluate the solution of these results we fitted our model over the training set and choose the λ's, performing best on the validation set. We look at the MSE as performance indicator to decide which result performs better.

(b)

In this part we use gradient descent to find the solution for the given problem. Again the λ's are chosen with a grid search in the same area as before. The gradient descent algorithm works as follows. We start with an initial β and update them as follows:

$$\beta_{k+1} = \beta_k - \alpha\nabla_\beta\hat{\beta}_{EN}(\beta_k)$$
$$\alpha = 0.2$$

Where β_k is the value of β at itteration k. We runs this until either the maximum amount of iterations is reached, or the change in the objective value is below a significance threshold (we choose 0.01).

(c)

Here, we use a pathwise coordinate descent algorithm (as described by J. Friedman et al (2007)). The lambdas are optimized as before. The β's are updated as follows:

$$\tilde{\beta}_j = \frac{S(\frac{\Sigma_{i=1}^n x_{ij}(y_i - \tilde{y}_i^{(j)})}{n}, \lambda_1)_+}{1 + \lambda_2}$$

Where:

$$\tilde{y}_i^{(j)} = \Sigma_{k \neq j} x_{ik} \tilde{\beta}_k(\lambda_1)$$

$$S(a.b) = \begin{cases} a - b, & if \quad a > 0 \quad and \quad b < |a| \\ a + b, & if \quad a < 0 \quad and \quad b < |a| \\ 0, & if \quad b \geq |a| \end{cases}$$

This runs for j = [1, 2, . . . p, 1, 2, . . .], until either the maximum amount of iterations is reached, or the change is below the significance threshold (here 0.000001).

(d)

The optimization problem for the binary programming formulation looks slightly different compared to the models mentioned before. Instead of penalizing the use of different independent variables, we straight up restrict to model to using a certain amount of independent variables. This restriction is set by the K parameter. The model looks as follows:

$$\begin{aligned} &\hat{\beta}_{MILP} = \arg\min_{\beta, z} && ||y - X\beta||_2^2 \\ &\text{Subject to} && \sum_{j=1}^{p} z_j \leq K, && K \leq p \\ & && -Mz_j \leq \beta \leq Mz_j, && M = \text{very large} \end{aligned}$$

We solve this problem using MATLAB with YALMIP and the GUROBI solver. Instead of looking for which λ performs well, we can look at which K the performs well.

Analysis

wTo analyze the different solving methods of the elastic net regularization we test the different solving methods for different data sets. We simulate the data sets in Pyton, and therefore a true model does exists for all the data. We generated 16 different kinds of data sets and looked at all the combinations of the following differences between the data sets:

	# observation (n)	# features (m)	# useful features (r)	amount of noise (ϵ)
values	100, 1000	30, 150	0.2m, 0.9m	2, 5

Table 1: Features

The noise value is the standard deviation of the error term added to the output.

To train our model and choose the right parameters for λ or K, we divide the data in a training, a validation and a test set. The validation set is for choosing the parameters and the test set is useful for comparing the methods with each other. We divided the data in 80-10-10 % for this. The results for the different methods can be found in the appendix.

Commercial solver
The first solving method does depends on the size of the problem. The method of using a commercial solver works pretty well, however it is pretty time consuming, the time significantly increases if the number of features increases, even with relatively few features it takes a lot of time. It also overfits the model a little bit more, sets less β's to zero, if the amount of noise increases. The biggest drawback of this method is the time, it does take a long time to solve the problem.

Gradient descent
The gradient descent method runs pretty fast. However, the performance time increases if both the number of features and the number of observations increases. It is less sensitive to changes in this than the commercial solver method, however if the problems becomes larger, the running time suffers. The accuracy of this method is good in cases were the number of observations (in the training set) is larger than the number of features. If the number of features becomes larger the performance becomes worse.

Pathwise coordinate descent
The path wise coordinate descent method outperforms all methods by far in time. The solving time is lower than a second, where it is uncommen for the first method to need more than 100 seconds. However, this method does need a lot of observations to train. Its performance in terms of MSE is a bit similar the the gradient descent method, however this method is 5 times faster.

Binary programming formulation
The binary programming formulation method fits the model pretty well. To train this data set we tested all possible $K's$ with a maximum solver time of 5 seconds for each, we continued with the current best solution. This is not that big of a problem because in most cases the GAP already dropped below 1%. This method fits the model well even in cases where the number of features outnumbers the number of observations. The biggest draw back of this method is that it has binary variables. Which means that if the number of features increases, the solving time explodes. Even if you know the right K to pick, solving a lot of binary variables costs a lot of time.

Conclusion

If we compare all the models, we see that if we have a lot of observations, they all converges to more or less the same solution. The pathwise method outperforms all other models in time in this case. Furthermore, the solving methods (and the general elastic net regularization), performs worse if there is more noise. The binary model and the commercial solver can deal with this a bit better than the algorithms, especially in cases with a lot of features compared to a lot of observations.

2 Client report

The problem is that we have an Elastic net regularization as discribed by (H. Zou and T. Hastie (2005)). This regression formula contains the basic with a both a Lasso and Ridge term. The Lasso term makes the solution more sparse, it does some feature selection. While the ridge term makes the solution more stable. This is a very useful problem however it can be a bit difficult to solve. Therefore we considered 4 methods to solve this problem:

1. First we used a commercial solver to solves this problem. We used MATLAB with YALMIP and GUROBI as solver for this problem.
2. Secondly the gradient descent method was used. In this method we start with a random feasible solution for the problem. We calculate the gradient of the Elastic net regularization to find in which direction a better solution for the problem can be found and move our point a bit towards that direction. This process is repeated until not progress is found.
3. We also solved the model using a pathwise coordinate descent algorithm as described by J. Friedman et al (2007). In this algorithm we again start with a random initial feasible point. However now, for each β coordinate, we move separately to the optimal solution. We first move the first β keeping the rest fixed and than the second keeping the rest fixed. After the last β we start again with the first and continue this process until no progress is made anymore.
4. In the last problem we put a hard constraint on the number of non-zero β's, instead of adding a penalty for it in the elastic net regularization. This new binary formulation problem is solved with the same solver as in the first method.

To look how these different methods perform, we looked at different sizes of data sets, different complexity within the data set and different quality of data set. By a complex data set we mean a data set that has a lot of variables in it. The quality of the data set is dependent on the amount of noise in the output.

An overview of the results can be found in the appendix. Here, we can see that any of the four models manages to converge. However, differences occur in performance and computation time. We will now analyze the models based on these differences and explain in which case a model should be used.

Commercial solver
Compared to the other models, this model works exceptionally well with small data sets. This can be very beneficial in situations where there is only a limited amount of data available within a complex model. An example of this case is when the data is linked to experiences. Say that a salesperson keeps track of the characteristics of the companies he or she tries to sell to. Based on these characteristics he or she wants to predict the chance of a successful sale to a new company. This can be a typical case where there is only limited data is available within a complex model. A drawback of this model is that the model take a lot of time in general. Also, the time it takes the model to run is dependent on the complexity and quality of the data set. If a data set is

more complex and has a lower quality, the running time of the model will increase rapidly. This means the model is very unsuitable for a fast-paced environment, situations where decisions need to be made quickly based on new data. A benefit of this model is that the time does not grow with the size of the data set. This is extremely beneficial for larger data sets. Especially within companies where a lot of data is available, this model may be very suitable. For instance if someone would want to predict the default rate of loans. This is a typical case where a lot of data is available. A word of caution however, would be that the model's running time is negatively impacted by the complexity of the model.

Gradient descent
Although this model performs very well in most situations, there are some situations where the model does not perform very well. The most important situation is where there is limited data available within a complex model. The salesperson example we mentioned before would be a case where the gradient descent model is unsuitable. Companies tend to have a lot of characteristics, making the model very complex. This in combination with the limited amount of data available, especially when starting out, will negatively impact the performance of this model. Time-wise the gradient descent model performs exceptionally well. In all cases analyzed, this model has a very low running time. This is highly beneficial in fast-paced environments. For instance in the financial markets, where it is very important to be able to react to very quickly changing conditions, the gradient descent model could be very suitable. Note however, the model's running time increases both with the size and complexity of the data set. This means the model might not be suitable for larger, complex data sets. An example here would be when working with time series. If the model considers a lot of historical data, the model is very complex. This negatively impacts the gradient descent algorithm.

Pathwise coordinate descent
This model performs similar to the gradient descent model. We again see that in the case where there is limited data available in a complex model, the model does not perform well compared to it's competitors. Again, the salesperson example can be an illustration of where this might become a problem. Also, as with the previous model, the pathwise coordinate descent model is a very fast model. In all cases we analyzed, it was one of the quickest models. Contrary to gradient descent, the model's running time does not increase with the amount of observations. This means the pathwise coordinate descent model is preferred over the gradient descent model very large data sets. Take for instance a data set of patients, which you want to use to predict the likelihood of someone getting a specific disease. If this data set contains a lot of patients, the pathwise coordinate descent model is likely to be faster than the gradient descent model. Lastly, the running time of this model does not suffer from an increase in complexity of the data set. Although this is a benefit, we have mentioned above that if the size of the data set is small, the model's performance might suffer.

Binary programming formulation
The binary programming formulation model performs very similar to the commercial solver. The biggest selling-point of this model is its ability to perform very well in a situation with a small, but complex, data set. It even manages to outperform the commercial solver in some cases. However, the model also suffers from the same drawbacks as the commercial solver. The binary

programming formulation takes a lot of time to compute. We see in many cases this method takes the longest out of all models. As with the commercial solver, the time also increases with the complexity of the model and the quality of the data. This means it is very unsuitable for fast-paced environments where decisions need to be made quickly, for in stance in the stock market. Finally, we do see that the time does not increase with an increase in the size of the data set. This means the model might be useful for very large data sets.

2.1 Conclusion

We have seen that the four models perform differently in different situations. We see that the commercial solver and the binary programming formulation perform very similar on a similar data set. Especially in the case where there is a relatively small, highly complex, and high quality data set, these models perform very well.

The gradient descent model and the pathwise coordinate descent model perform very similar as well. We see that these models perform very similar to the other two models, while achieving a much lower running time. There is one exception however, in the case of a small, highly complex data set, the gradient descent and pathwise coordinate descent model underperform compared to the other two models. In this case, one of the other two models is preferred.

2.2 Advice

We differentiate two situations for our advice. The first situation is a situation where there is a limited amount of data available within a complex data set. In this situation we advice to use the commercial solver method. As we have explained, together with the binary programming formulation, it outperforms the other two models. Considering its time advantage over the binary programming formulation, we prefer this model.

In all other cases, we see that all four models achieve a very similar error rate. Considering that out of all four models, the pathwise coordinate descent model is by far the fastest, we advice to use this model in all these situations.

3 Appendix

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	53,50	57,09	266,53	0,01	0,01	-	12,00
Gradient	53,31	58,29	0,53	0,01	0,01	-	12,00
Pathwise	58,25	50,35	0,08	0,00	0,00	-	14,00
Binary	48,84	27,18	196,56	-	-	11,00	19,00

Table 2: $n = 100, m = 30, r = 27, \epsilon = 5$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	25,29	25,40	264,66	0,10	0,01	-	23,00
Gradient	25,53	25,35	1,01	0,10	0,01	-	23,00
Pathwise	25,31	25,64	0,18	0,00	0,10	-	23,00
Binary	24,90	25,49	188,68	-	-	11,00	23,00

Table 3: $n = 1000, m = 30, r = 27, \epsilon = 5$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	61,46	13,13	373,50	0,10	0,00	-	84,00
Gradient	124,66	64,73	2,22	1,00	0,00	-	129,00
Pathwise	119,33	66,93	0,23	1,00	0,00	-	142,00
Binary	36,63	17,03	1513,85	-	-	7,00	143,00

Table 4: $n = 100, m = 150, r = 135, \epsilon = 5$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	28,06	28,37	480,78	0,10	0,01	-	84,00
Gradient	27,83	28,94	10,72	0,10	0,01	-	77,00
Pathwise	27,10	28,20	0,68	0,10	0,01	-	104,00
Binary	24,96	27,97	2208,42	-	-	6,00	144,00

Table 5: $n = 1000, m = 150, r = 135, \epsilon = 5$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	51,78	25,64	487,25	0,10	0,01	-	12,00
Gradient	51,10	23,83	0,26	0,10	0,00	-	13,00
Pathwise	43,64	19,03	0,08	0,01	0,00	-	19,00
Binary	38,11	27,82	361,01	-	-	8,00	22,00

Table 6: $n = 100, m = 30, r = 6, \epsilon = 5$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	25,36	26,81	504,73	0,01	0,00	-	26,00
Gradient	25,25	26,89	0,54	0,00	0,00	-	26,00
Pathwise	25,72	26,74	0,19	0,00	0,01	-	26,00
Binary	25,07	26,75	372,15	-	-	18,00	26,00

Table 7: $n = 1000, m = 30, r = 6, \epsilon = 5$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	52,10	60,81	599,42	1,00	0,00	-	143,00
Gradient	59,93	79,15	1,46	0,10	0,01	-	115,00
Pathwise	57,93	65,62	0,31	0,10	0,00	-	136,00
Binary	36,79	88,81	2432,00	-	-	3,00	147,00

Table 8: $n = 100, m = 150, r = 30, \epsilon = 5$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	26,88	27,61	130,72	0,10	0,10	-	91,00
Gradient	27,13	28,07	9,04	0,00	1,00	-	57,00
Pathwise	26,93	27,08	1,08	0,10	1,00	-	123,00
Binary	25,61	26,56	901,49	-	-	14,00	136,00

Table 9: $n = 1000, m = 150, r = 30, \epsilon = 5$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	8,58	9,32	16,83	0,01	0,00	-	19,00
Gradient	8,61	10,42	0,35	0,00	0,00	-	18,00
Pathwise	12,51	10,71	0,12	0,00	0,00	-	20,00
Binary	6,70	3,85	9,45	-	-	10,00	20,00

Table 10: $n = 100, m = 30, r = 27, \epsilon = 2$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	4,09	4,12	16,98	0,01	0,01	-	23,00
Gradient	4,20	4,23	0,90	0,01	0,00	-	23,00
Pathwise	4,09	4,10	0,14	0,01	0,01	-	23,00
Binary	4,00	4,07	9,98	-	-	11,00	23,00

Table 11: $n = 1000, m = 30, r = 27, \epsilon = 2$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	9,23	2,07	118,35	0,10	0,00	-	108,00
Gradient	45,90	43,98	1,68	0,10	0,00	-	74,00
Pathwise	82,63	77,49	0,27	1,00	0,00	-	143,00
Binary	5,68	2,97	629,68	-	-	9,00	141,00

Table 12: $n = 100, m = 150, r = 135, \epsilon = 2$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	4,35	4,54	127,42	0,10	0,00	-	132,00
Gradient	4,51	4,93	12,37	0,10	0,00	-	122,00
Pathwise	4,26	4,54	0,66	0,10	0,00	-	136,00
Binary	4,10	4,23	850,09	-	-	9,00	141,00

Table 13: $n = 1000, m = 150, r = 135, \epsilon = 2$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	7,62	3,79	17,04	0,10	0,00	-	12,00
Gradient	8,66	4,97	0,33	0,10	0,00	-	6,00
Pathwise	7,45	3,05	0,14	0,01	0,00	-	14,00
Binary	6,06	3,26	10,39	-	-	9,00	21,00

Table 14: $n = 100, m = 30, r = 6, \epsilon = 2$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	4,06	4,29	17,20	0,01	0,00	-	12,00
Gradient	4,06	4,35	0,86	0,01	0,00	-	14,00
Pathwise	4,14	4,30	0,19	0,00	0,00	-	18,00
Binary	4,04	4,36	10,40	-	-	13,00	17,00

Table 15: $n = 1000, m = 30, r = 6, \epsilon = 2$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	9,10	14,05	115,86	0,10	0,00	-	115,00
Gradient	25,14	31,26	1,98	0,10	0,00	-	73,00
Pathwise	31,24	35,13	0,30	0,10	0,00	-	106,00
Binary	6,34	13,24	619,57	-	-	14,00	136,00

Table 16: $n = 100, m = 150, r = 30, \epsilon = 2$

	MSE val	MSE test	Time (s)	λ_1	λ_2	K	Zero-β
Base	4,33	4,26	130,80	0,10	0,00	-	135,00
Gradient	4,69	4,46	11,07	0,10	0,00	-	125,00
Pathwise	4,33	4,28	0,97	0,10	0,00	-	137,00
Binary	4,11	4,21	896,30	-	-	9,00	141,00

Table 17: $n = 1000, m = 150, r = 30, \epsilon = 2$