Sparse Least Square Linear Regression

Introduction

Recently a LocalSolver customer has to model a linear regression model with additional combinatorial constraints. The mathematical formulation of the problem was quite easy but there are at least three models that can be implemented in LocalSolver, from a simple continuous optimization problem to a more complex binary one using advanced features. The fastest model is nearly 10 time faster than the others to find similar solution. This article describes the three proposed model on a sparse linear regression model with one combinatorial constraint and discuss the advantages and drawbacks of each model. If you are already familiar with linear regression and sparse linear regression you can skip the first two paragraphs.

Linear Regression

Linear Regression is an approach to predict the behavior of a variable using a linear combination of explanatory variables (1). For instance, one can model the price of a car according to its size, its weight, its consumption, its power and so on. To build the model you need to have several examples of cars with their price and their explanatory variables. More formally assume our dataset is composed of m samples and let $Y \in \mathbb{R}^m$ be the vector that we want to explain with a linear combination of n explanatory variables $X \in \mathbb{R}^{m \times n}$. The linear model is described by a vector $\beta \in \mathbb{R}^n$ of parameters that satisfy the following system of equations $Y = X\beta$.

This system could be overdetermined or underdetermined and in practice we pick the parameters β that is the solution of the following optimization problem:

$$\min_{\beta} \|Y - X\beta\|_2$$

This continuous optimization problem is well studied and there is an analytical formula to find the optimal parameters.

Sparse Linear Regression

In practice the number of explanatory variables can be very high and most of these variables are not related to the one we want to predict. In our car price example, some explanatory variables can represent the length of the antenna, the wavelength of the body paint and it is very unlikely that these variables can be linearly linked to the price of the car. Variables with low explanatory power must be discarded from the model as they can lead to some overfitting and make more complex the model readability. To overcome this issue, one can try to find a sparse model with only a few non zeros parameters.

The classical approaches to sparse linear regression add a regularization term in the objective function. In the Lasso model the regularization term is a L_1 penalization of the parameters and in the Ridge regression approach the regularization is a L_2 penalization. For instance, the Lasso model solves the following optimization problem for a given value of λ:

$$\min_{\beta} \|Y - X\beta\|_2 + \lambda\|\beta\|_1$$

Another approach is to add combinatorial constraints to the model to restrict the structure of the solutions (3). In particular we are interested in restricting the vector of parameters to have less than a

given number of non zeros entries (the L_0 norm). This problem is NP-Hard (2) but good solutions can be found in practice using mixed integers quadratic solvers or heuristics.

LocalSolver models

The Sparse Linear regression model that we want to solve is the following Mixed Integers Quadratic optimization problem:

$$\min_{\beta} \|Y - X\beta\|_2$$
$$st. \|\beta\|_0 \leq k$$

This mathematical optimization problem can be solved using a purely continuous model, a mixed integers model or a purely binary model.

Continuous model

A first attempt to model this problem is to use only continuous decisions and to create expressions to compute the number of non zeros entries in beta. Here is our LocalSolver model using the Python API:

```
RANGE=10
ls = localsolver.LocalSolver()
model = ls.model
beta = [model.float(-RANGE, RANGE) for j in range(n)]
nzeros = [beta[j] != 0.0 for j in range(n)]
totalError = model.sum()
for i in range(m):
        rowValue = model.sum([beta[j] * X[i][j] for j in range(n)])
        rowErr = rowValue - Y[i]
        totalError.add_operand(rowErr * rowErr)
model.add_constraint(model.sum(nzeros) <= k)

model.minimize(totalError)
model.close()
```

$$\min_{\beta} \|Y - X\beta\|_2$$
$$st. \sum_j \beta_j \neq 0 \leq k, \forall i$$

Using a random instance with 1000 samples and 259 explanatory variables, this model is not able to find a feasible solution in 60s. The first problem is a numerical precision issue with the comparison between the parameter value and zero. The second problem is that 0 is a special value in the domain that should be handled specifically. If LocalSolver starts from a feasible point (all the parameters values are 0), it is able to find a solution of cost 14482.

Mixed Integer model

To avoid the special value 0 in the variable range, let us create new binary decisions that decides if a parameter value is zero or not.

```
RANGE=10
ls = localsolver.LocalSolver()
model = ls.model
beta = [model.float(-RANGE, RANGE) for j in range(n)]
nzeros = [model.bool() for j in range(n)]
totalError = model.sum()
for i in range(m):
        rowValue = model.sum([nzeros[j] * beta[j] * X[i][j] for j in range(n)])
        rowErr = rowValue - Y[i]
        totalError.add_operand(rowErr * rowErr)
model.add_constraint(model.sum(nzeros) <= k)

model.minimize(totalError)
model.close()
```

$$\min_{\beta, z\in\{0,1\}^n} \|Y - X\beta'\|_2$$
$$st. \begin{cases} \sum_j z_j \leq k, \forall i \\ \beta'_j = z_j\beta_j, \forall j \end{cases}$$

With this model the solver is able to find a feasible solution in less than a second but after 60s the solver has a solution of only 29018. The main problem with this alternative model is that we broke the "locality" between two solutions. In order to move from one feasible solution to another with a close objective value we need to change the value a binary decision and a continuous one.

Binary model

To avoid the "locality" problem, recall that once the non-zero decisions are fixed, finding the parameters is just an ordinary least square problem on the non-zero dimension that can be easily found using linear algebra. With LocalSolver native function operator, we can call a linear algebra solver to find the optimal parameters during the search.

```
ls = localsolver.LocalSolver()
model = ls.model
nzeros = [model.bool() for j in range(n)]

#call external linear solver in numpy
totalErrorEvaluator = model.create_native_function(native_leastsq)
totalError = model.call(totalErrorEvaluator)
totalError.add_operands(nzeros)

model.add_constraint(model.sum(nzeros) <= k)

model.minimize(totalError)
model.close()
```

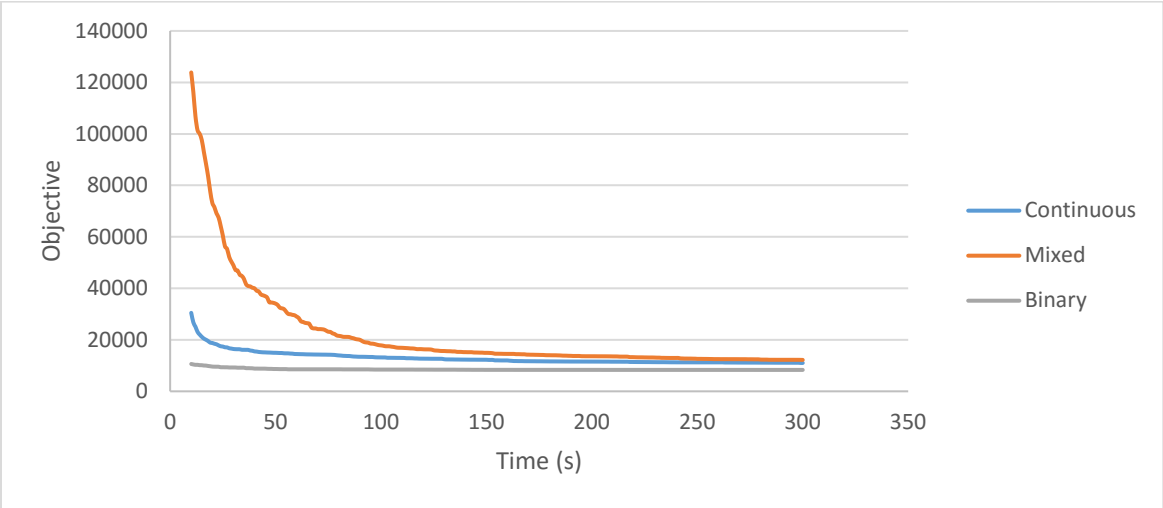
$$\min_{z\in\{0,1\}^n} \|Y - X\beta\|_2$$

$$st. \begin{cases} \sum_j z_j \leq k, \forall i \\ \beta = argmin_\gamma \|Y - X_z\gamma\|_2 \end{cases}$$

Here in less than 10 seconds we are able to find a solution with a cost of 10622.7 even if the number of iteration per seconds is reduced by calling the linear algebra solver.

Conclusion

The sparse linear regression problem can be tackled by at least 3 different LocalSolver models. The performance differences between the three models are illustrated by their convergence curves (values before 10s are not shown to increase readability):



The continuous model seems to be the more natural but has some trouble handling combinatorial constraints. The introduction of binary decision is helpful to handle the combinatorial constraints but the convergence rate is worse than the continuous model. Finally, the last model uses only binary decision to handle the combinatorial constraint and decides the parameters values by solving linear algebra problem. This last model is able to find good solutions in faster than the other models.

References

1. **Linear Regression. *Wikipedia.* [Online] 2016. https://en.wikipedia.org/wiki/Linear_regression.**

2. Welch, W. Algorithmic complexity: three NP-hard problems in computational statistics. *Journal of Statistical Computation and Simulation.* 1982, Vol. 15, 1.

3. D. Bertsimas, A. King. An Algorithmic Approach to Linear Regression. *OPERATIONS RESEARCH.* 2016, Vol. 64, 1.