Comparing Methods of Variable Selection for Sparse High-Dimensional Data Sets

Justin Bensley

December 2022

1 Summary

This project aims to compare various methods of variable selection for large complex data sets with very few meaningful variables. These high-dimensional sparse data sets are incredibly taxing for traditional methods of variable selection such as forward or backward elimination as there is simply too many variables to step through and consider. The data set that was chosen for this was the clean FIFA 2018 data set. This is a large data set containing roughly 65 different variables for thousands of different players. While sparse high-dimensional data sets can get much much larger than this this data set was chosen to be reasonable with run times and trial and error. The regression was used to predict a player's overall rating out of one hundred based on the roughly 65 variables. The algorithms were run in python and R, both through google colab. There were two main groupings of algorithms; the LASSO method and its variations and methods tied to *Sparse High Dimensional Regression: Exact Algorithms and Phase Transitions*. We start by looking at the LASSO methods. We first run the basic LASSO method to see how well it performs. The basic LASSO had the quickest run time at .123772 seconds and had a solid R^2 of .92977, but it should be noted that this value of $\lambda = 1$ was chosen after trial and error. In a setting where everything is truly unknown this method would be far more unreliable due to the lack of knowing our constraint λ. The first special LASSO that was implemented was the adaptive LASSO, which works by giving the LASSO pre-existing weights to work with that allow it to run without worry of over penalizing the data set. This method did not perform particularly well in these trials, sporting the highest run time of 17.350 seconds and the lowest R^2 of .9259. The last LASSO method was a combination method of LASSO and forward elimination. This works by setting a low penalizing λ to the LASSO and running it as normal. This clears a major bulk of the data out but still leaves considerable room for trimming variables. From there the classical methods of forward or backward elimination are applied to find the most important variables. This benefits from the LASSO's quickness for removing the bulk of the variables but also has the finesse required to not remove too many. This method had a run time of 5.380594 seconds while having a R^2 of .92661. This method ran slower than the regular LASSO due to the forward elimination used, but ran considerably faster than adaptive LASSO. Additionally it sported a better R^2 than the adaptive LASSO. Overall if a LASSO method must be used I would use this one. Next we dive into the methods tied to *Sparse High Dimensional Regression: Exact Algorithms and Phase Transitions*. This is an important comparison to the LASSO methods, as the paper disparages the use of LASSO. The first of these methods is L0Learn, which works to solve L0 regularization through coordinate descent to find optimal λ values and variables. This method was also incredibly quick, averaging .447278 seconds with an R^2 of .9287. This method was faster than the two specialized LASSOs and provided a better R^2, suggesting that the paper may have had a reason to have gripes with the LASSO. The last method looked at was the ABESS method. This package utilizes a combination of splicing and L2 regularization to perform

variable elimination. ABESS had an average run time of 2.207968 seconds and a R^2 score of .93013. This is what I would consider the best method to choose from the 5. It had a faster run time than the specialized LASSOs and had the best R^2 score out of any algorithm, which of course leads to the best possible model. Overall the two processes with connections to *Sparse High Dimensional Regression: Exact Algorithms and Phase Transitions* outperformed the LASSO based models, which gives credit to the strength of the algorithms proposed by the paper.

2 Background

The most important part of this project is the data set that was chosen. I chose a reasonable data set that could be viewed as a sparse data set, the cleaned FIFA 2018 football data set. This covers roughly 70 different aspects for thousands of players within the FIFA organization. To begin I chose what would be my dependent y variable and what variables would be independent. In this case I chose overall rating as the dependent variable, meaning our regression occurs to try to predict a players overall rating from the independent variables. Additionally, I pruned a handful of variables from the data set from the start such as: nationality, name, club, etc. I did this for the simple reason of eliminating non numerical variables. The only numerical variable I eliminated was yearly value, as I wanted to focus on the total salary instead. This left 64 columns of independent variables and one column for the dependent y. The next important thing to discuss is the implementation of this project. The code was primarily written in python, making use of pandas for data processing, along with numpy, sklearn and asgl for variable selection. Sklearn was used for implementation of the LASSO function while asgl was used for the implementation of the adaptive LASSO. Additionally, the python package ABESS was used to perform the best subset selection method. A portion of the code was also written in R, for the implementation of L0Learn. This package attempts to find solutions to the L0 norm, much like the Bertsimas paper. Lastly, an attempt was made to implement the cutting plane method described by the paper *Sparse High Dimensional Regression: Exact Scalable Algorithms and Phase Transitions*. Unfortunately, many issues arose with implementing the package in Julia. The reason these methods are used over the traditional techniques of backward/forward elimination is due to the large run times that would be required when dealing with the bloated amount of variables in the data sets.

3 Methods

To begin we look at the LASSO and its variants, then move to the L0 method used by L0Learn before finishing off at ABESS. Additionally, there is a dive into a method that can be used for non-linear circumstances. I think its important to note when sparse high-dimensional regression is to be used. The sparsity in

this case doesn't refer to a set of 1's and 0's, it refers to the important and unimportant parts of a data set with an incredibly large number of variables. The term "high-dimensional" is also referring to the data set, in this case the before mentioned incredibly large number of variables. Thus together we see the sparse high-dimensional regression is regression on an incredibly large data set in which only parts are useful. The main work of this project is based on high-dimensional regression, specifically the methods described in *Sparse High-Dimensional Regression: Exact Scalable Algorithms and Phase Transitions*. While this method is offering an alternative to LASSO (least absolute shrinkage and selection operator), we need to form a point of comparison so LASSO will also be inspected. Since LASSO seems to be the preferred method of sparse regression it would be prudent to start with how it works and the methods surrounding it.

The LASSO method can be thought of as linear regression with a little extra spice. In normal linear regression the estimator $\hat{\beta}$ is the one that minimizes the difference between the true Y and the predicted Y: $x * \hat{\beta}$, or in more mathematical terms, we seek the estimator that minimizes:

$$\Sigma Y_i - x * \hat{\beta}$$

While this works for models with low variances, those with high variances will find that the model may not work well for general data sets. The LASSO aims to solve this problem by adding in a penalty term to the equation above, so LASSO estimators minimize the equation:

$$\Sigma Y_i - x\hat{\beta} + \lambda\Sigma|\hat{\beta}_j|$$

Where λ is the penalty term. As one can see LASSO can easily become OLS regression by setting $\lambda = 0$. However, as λ grows our penalizing term causes unhelpful variables to be reduced down to zero when the minimization occurs! It is important to note then, if λ starts to head towards ∞, then one will find that all the values of $\hat{\beta}$ will go towards 0, as they will not contribute since λ dominates the sum regardless. Why then is this important for sparse high-dimensional regression? Well as mentioned in the beginning these data sets are large and filled with useless variables. LASSO not only returns a fitted regression model, but also sets the coefficients with useless parameters to 0. In this sense LASSO performs both regression and variable elimination in the same step. The paper *Adaptive Lasso For Sparse High-Dimensional Regression Models* describes a variation of the LASSO method called the adaptive lasso. In this scenario there is a pre-existing estimator that can be used to create weights for our estimator. Per *Adaptive Lasso For Sparse High-Dimensional Regression Models*, the weights and the equation to be minimized our given by:

$$w_j = |\beta_p|^{-1}$$
$$\Sigma(Y_i - x_i\beta)^2 + \lambda\Sigma w_j * |\beta_j|$$

Where β_p is the pre-existing estimator. The adaptive LASSO gains the benefit of gaining the oracle property, that is it identifies the nonzero coefficients with increasing probability that approaches one and the estimators of those coefficients are asymptotically normal. Since the adaptive LASSO produces oracle estimators, it produces better results for variable elimination than the basic LASSO. Thus optimal use of a LASSO estimator is to be used as a weight for an adaptive LASSO estimator.

Furthermore, *Hierarchical selection of variables in sparse high-dimensional regression* suggests a multiple step approach as well. This method makes use of LASSO as well, however notes that LASSO is prone to removing to many variables. Thus the paper suggests a multi-step hierarchical model, in which the initial step is to perform LASSO to get the first subgroup. This method does not use LASSO to find all of the selected variables, but instead uses a lower λ to weed out less variables. The final outcome of this first LASSO will be a set of variables that is substantially smaller than the initial set, but also bigger than the optimal choice. This leads to the second step, which is to treat the new smaller set of variables as a regular regression model and perform backward/forward selection to remove the remaining variables. Once the final model is completed the steps repeat with the inclusion of interaction terms until a lack of change in the testing variable occurs. The text suggests the use of R^2 for this testing variable over others such as AIC. This method addresses one of the weaknesses of LASSO: its hard to know how many variables are important and thus hard to determine an appropriate λ.

Now that alternative methods have been described its important to note *Sparse High-Dimensional Regression: Exact Scalable Algorithms and Phase Transitions* describes the issues with LASSO, that is the data struggles to be recovered after sparsity. They propose a solution involving phase transitions that make the sparsity recoverable. The main theorem in this paper involves transforming the sparse regression problem into a nonlinear optimization problem to be minimized below:

$$\min \tfrac{1}{2} Y^T (I_n + \gamma \Sigma s_j K_j)^{-1} Y$$

with:

$$K_j = X * X^T$$

To solve this they recommend the application of a cutting plane algorithm. The idea behind this is to continuously cut of parts of the binary solution until an optimal solution is found. This optimal solution is then used to find the solution to the sparse regression problem.

The next algorithm that I read about is the method applied in L0Learn. This is a L0 regularized method, that seeks to minimization the equation:

$$\frac{1}{2}||y - X\beta||_2^2 + \lambda||\beta||_0$$

The algorithm that runs this process implements a form of coordinate descent. This works by creating a grid of possible λ values based on the input data. This grid is then stepped through finding an optimal path to the solution set of regularization terms. This allows the process to be done quickly, even though most cases of L0 regularization methods are slow.

The last algorithm that I applied for this project involved the use of the ABESS package. This package was built to find the best subset of variables using a few different techniques. It starts by performing splicing,then it groups the variables together and lastly performs L2 regularization as described in *Sparse High-Dimensional Regression: Exact Scalable Algorithms and Phase Transitions*, meaning this package is a shown application of the paper!

But what if the data doesn't follow a linear fit? In this case a non-negative garrote is applied, as explained by *Variable Selection for Sparse High-Dimensional Nonlinear Regression Models by Combining Nonnegative Garrote and Sure Independence Screening*. The format of the non-negative garrote looks very similar to that of the LASSO. In the case of the non-negative garrote, instead of minimizing $\hat{\beta}$ it minimizes c_j in the equation:

$$.5\Sigma(Y_i - \Sigma c_j X_i \beta)^2 + n\lambda\Sigma c_j$$

For this to be a non-negative garrote the condition of having c_j be positive for all j is placed on the minimizer. With these garrote values a garrote estimate can be found,$\beta = c_j * \hat{\beta}$, where $\hat{\beta}$ is the whole model fit. Examining the equation to be minimized, one can see the similarities to LASSO, for large λ we get a sparse high-dimensional data set. For the nonlinear case the nonlinear regression model is fit to obtain initial parameters. Those initial parameters are then fed into the garrote equation, in which the garrote variables are returned. The garrote variables then create a new estimate for β, which can then be used as initial variables until convergence occurs. This process is relatively straight forward, and can be applied to high-dimension data sets for variable elimination in a non linear model.

4 Results

As mentioned previously the results were found using both python and R. To determine which algorithm was quickest I ran them each 5 times and found the average run time in seconds for each. The purpose of this was to examine whether any of the algorithms could be found to be slower, and thus easily ranked lower than the others. The table below shows the comparison between the five methods.

	RUN 1	RUN 2	RUN 3	RUN 4	RUN 5	AVERAGE RUN
LASSO	0.12376	0.12897	0.1167	0.11878	0.13065	0.123772
ADAPTIVE LASSO	16.84948	17.04554	17.58384	17.67764	17.59526	17.350352
LASSO + FORWARD ELIMINATION	5.45146	5.14274	4.85711	5.38418	6.06748	5.380594
L0LEARN	0.37039	0.66503	0.53175	0.32362	0.3456	0.447278
ABESS	2.39058	2.08137	2.18455	2.32462	2.05872	2.207968

Here we see that the quickest run program is the LASSO method, which takes under a tenth of a second to run. The quickness of the LASSO is not a surprise, as this process isn't precise, and relies on the choice of λ to produce results. The next fastest was L0LEarn, which typically took half a second to run. This method returns a list of λ values and the user chooses the corresponding lambda for the number of variables desired. The middle speed for the five algorithms belongs to ABESS, which typically took a little over two seconds to run. This is a considerable time jump compared to L0Learn and LASSO, but it must also be noticed that ABESS and the next two algorithms choose the optimal number of variables to include in addition to the best variables. Next came the LASSO and forward elimination combination, taking a little over 5 seconds to run. It can be seen that the methods are starting to get rapidly slower! The last and slowest method is the adaptive LASSO. This method took over 17 seconds to run, as well as requiring some initial information regarding possible λ values.
The next important result to inspect is the variables selected. It is also necessary to recognize at this point which variables are constant across the 5 methods and which aren't, as major discrepancies in that regard could be cause for concern. The table below outlines the number of shared, unique and total variables that each method produced.

	SHARED	UNIQUE	TOTAL
LASSO	18	0	18
ADAPTIVE LASSO	17	19	36
LASSO + FORWARD ELIMINATION	11	0	11
L0LEARN	14	0	14
ABESS	19	0	19

We can see that only the adaptive LASSO had unique values, which makes sense as it selected the largest number of variables. Most of the algorithms selected between ten and twenty variables of the sixty-four that were important. This is a huge decrease in the data set which makes it much easier to work with. The most important thing to look at here though is those variables that all five algorithms selected, there were nine such variables. Those nine variables were: age, ball control, composure, heading accuracy, positioning, potential, reactions, sprint speed, and strength. Since these are the nine variables that all algorithms agreed upon it is safe to assume that these variables are always going to be important when looking at a players overall score.

The last thing to look at is how well each model fits the data. This is done by taking the variables that each model selected and fitting a linear regression model with sklearn and examining the R^2 score that is returned. The R^2 for each model is listed below:

	R2 SCORE
LASSO	0.92977
ADAPTIVE LASSO	0.9259
LASSO + FORWARD ELIMINATION	0.92661
L0LEARN	0.9287
ABESS	0.93013

The first thing to notice is that all of the models roughly provide the same R^2 value meaning any of the methods used would be more than adequate for variable selection. That being said the adaptive LASSO provided us with the lowest R^2 score while ABESS provided the highest score. While its only an increase of .000423, it does this while including almost two times the amount of variables. Thus it is safe to assume that the adaptive lasso chose too many variables and was appropriately penalized. Another thing to notice is that the R^2 value increases with the variables added, except for the adaptive lasso, so the models which chose more variables had a higher scored. Thus we can make the assumption that ABESS chose an optimal point in which to stop adding variables.

5 Conclusions

Each of the models was able to generate an adequate R^2 score of roughly 92.8%. That is to say that each model was able to successfully trim the large complex data set down to a vastly smaller and easier to work with data set that isolated the key variables for regression. That being said each method was not equal in performance or output, and each model has an appropriate time and place. For instance, the basic LASSO method scored very well when it came to run time and R^2, but this was mostly due to a lucky choice of λ. I had tried a few different values of λ before settling on $\lambda = 1$, and those had drastically different variable selections. So while the choice of λ that I chose worked well its important to note that LASSO is like taking a sledge hammer to the problem, you may get lucky and break the data into the right size but more often than not you'll blindly chop away at the data cutting either too much or not enough out. There are a few solutions to this problem, the two discussed in this project are through the use of the adaptive LASSO or combining LASSO with other popular variable selection methods such as forward elimination. The adaptive LASSO in this case struggled to eliminate many variables and provided the longest run time and lowest R^2. On the other hand the mixture of LASSO and forward

elimination provided a solid model with a reasonable run time that also didn't run the risk of eliminating too many variables as in the regular LASSO. The last two models run had relation to the paper I chose, *Sparse High-Dimensional Regression: Exact Scalable Algorithms and Phase Transitions*, and thus will be more critically compared to the LASSO methods. Starting with L0Learn, this model provided an excellent run time and a stellar R^2 value. It provided various λ values and told which to choose depending on how many variables I wanted. I thought this method was better than the LASSO methods as it was quick, painless to use and its output provided options instead of guesses. While this didn't initially give the best subset it was easy enough to find by just comparing the R^2 values for the various λs that were provided to me by the package. The final model was the one that I thought ran the best, the ABESS algorithm. It ran slower than L0Learn, but also provided the highest R^2 value. If the run time was considerably longer there would be more concern here but it still ran relatively quick while providing the best subset possible, without any guess work like in L0Learn. Going forward if I had to learn more about this subject I would dive deeper into the nonlinear regression aspects of sparse data sets, earlier I had described an approach on how to do this and I have to wonder whether there are other options for this out there. Another thing I would approach is the adaptive LASSO. While it did not perform well here I remain unfazed and believe it should work better than the regular LASSO. Overall I am happy with the models and analysis I constructed and pleased to have created a successful model for predicting a player's overall score based on the data provided. Lastly, I would say I agree with *Sparse High-Dimensional Regression: Exact Scalable Algorithms and Phase Transitions* in that the LASSO method has issues that need to be addressed, while the cutting plane and other non LASSO methods negate those issues while running quickly and providing an accurate model.

6 Bibliography

Abess-Team. "Abess-Team/ABESS: Fast Best-Subset Selection Library." GitHub, https://github.com/abess-team/abess.

"ASGL." PyPI, https://pypi.org/project/asgl/.

Bertsimas, Dimitris, and Bart Van Parys. "Sparse High-Dimensional Regression: Exact Scalable Algorithms and Phase Transitions." The Annals of Statistics, vol. 48, no. 1, 2020, https://doi.org/10.1214/18-aos1804.

Bickel, Peter J., et al. "Hierarchical Selection of Variables in Sparse High-Dimensional Regression." Institute of Mathematical Statistics Collections, 2010, pp. 56–69., https://doi.org/10.1214/10-imscoll605.

Civieta, Álvaro Méndez. "An Adaptive Lasso." Medium, Towards Data Science, 17 Aug. 2020, https://towardsdatascience.com/an-adaptive-lasso-63afca54b80d.

Fan, Jianqing, et al. "Sparse High Dimensional Models in Economics." SSRN Electronic Journal, 2010, https://doi.org/10.2139/ssrn.1659322.

"Feature Selection Using Selectfrommodel and LASSOCV¶." Scikit, https://scikit-learn.org/0.22/auto$_e$$xamples/f$$eature_s$$election/plot_s$$elect_f$$rom_m$$odel_b$$oston.html$.

Hazimehh. "Hazimehh/l0learn: Efficient Algorithms for L0 Regularized Learning." GitHub, https://github.com/hazimehh/L0Learn.

"High Dimensional Robust Sparse Regression." DeepAI, 29 May 2018, https://deepai.org/publication/high-dimensional-robust-sparse-regression.

Huang, Jian, et al. "Adaptive Lasso for Sparse High-Dimensional Regression Models." PolyU, Institute of Statistical Science, 1 Oct. 2008,

https://research.polyu.edu.hk/en/publications/adaptive-lasso-for-sparse-high-dimensional-regression-models.

L0Learn: A Scalable Package for Sparselearningusingl0 ... - ArXiv. https://arxiv.org/pdf/2202.04820.pdf.

Li, Yanfang, and Jinzhu Jia. "L1 Least Squares for Sparse High-Dimensional LDA." Electronic Journal of Statistics, vol. 11, no. 1, 2017, https://doi.org/10.1214/17-ejs1288.

Ning, Yang, and Han Liu. "A General Theory of Hypothesis Tests and Confidence Regions for Sparse High Dimensional Models." The Annals of Statistics, vol. 45, no. 1, 2017, https://doi.org/10.1214/16-aos1448.

Wu, Shuang, et al. "Variable Selection for Sparse High-Dimensional Nonlinear Regression Models by Combining Nonnegative Garrote and Sure Independence Screening." Statistica Sinica, 2014, https://doi.org/10.5705/ss.2012.316.

Zhu, Jin, et al. "Abess: A Fast Best Subset Selection Library in Python and R." ArXiv.org, 17 June 2022, https://arxiv.org/abs/2110.09697.

7 Appendix

```
#The Python code starts here until the R code starts



import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso, LogisticRegression , LinearRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import SequentialFeatureSelector
!pip install asgl
import asgl
import time


#load in data
dat=pd.read_csv('fifa18_clean.csv')



#remove non numerical columns
dat=dat.drop(['Name','Photo','Nationality','Flag','Club','Club Logo','Wage (€)','Preferred Positions','ID'],axis=1)
dat=dat.dropna()
dat.head()
```

	Value (€)	Age	Overall	Potential	Special	Acceleration	Aggression	Agility	Balance	Ball control	...	RB	RCB	RCM	RI
0	95500000	32	94	94	2228	89.0	63.0	89.0	63.0	93.0	...	61.0	53.0	82.0	62
1	105000000	30	93	93	2154	92.0	48.0	90.0	95.0	95.0	...	57.0	45.0	84.0	59
2	123000000	25	92	94	2100	94.0	56.0	96.0	82.0	95.0	...	59.0	46.0	79.0	59
3	97000000	30	92	92	2291	88.0	78.0	86.0	60.0	91.0	...	64.0	58.0	80.0	65
5	92000000	28	91	91	2143	79.0	80.0	78.0	80.0	89.0	...	58.0	57.0	78.0	62

5 rows × 65 columns

```
#split the dataframe into variables and target
targ=dat['Overall']
var=dat.drop('Overall',axis=1)


targ.to_csv(r'/content/target.csv',index=False,header=True)
var.to_csv(r'/content/var.csv',index=False,header=True)


#perform LASSO

ts=time.time()
las=SelectFromModel(Lasso(alpha=1,random_state=10))
las.fit(var,targ)
a=las.get_support()
lv=var.drop(np.array(var.columns[(las.estimator_.coef_==0).ravel().tolist()]),axis=1)
tss=time.time()
print(tss-ts)
```

```
0.11608600616455078
```

```
lv.head()
```

	Value (€)	Age	Potential	Special	Acceleration	Balance	Ball control	Composure	Free kick accuracy	Heading accuracy	Long shots	Positioning	Re
0	95500000	32	94	2228	89.0	63.0	93.0	95.0	76.0	88.0	92.0	95.0	
1	105000000	30	93	2154	92.0	95.0	95.0	96.0	90.0	71.0	88.0	93.0	
2	123000000	25	94	2100	94.0	82.0	95.0	92.0	84.0	62.0	77.0	90.0	
3	97000000	30	92	2291	88.0	60.0	91.0	83.0	84.0	77.0	86.0	92.0	

```
#perform adaptive lasso
```

```
ts=time.time()
alasso = asgl.TVT(model='lm', penalization='alasso', lambda1= .001*np.arange(0, 10, 0.1),parallel=True,
                      weight_technique='lasso', error_type='MSE', random_state=1)
alasso_result = alasso.train_validate_test(x=var.to_numpy(), y=targ.to_numpy())
print(time.time()-ts)
```

```
/usr/local/lib/python3.8/dist-packages/cvxpy/problems/problem.py:1337: UserWarning: Solution may be inaccurate. Try another solver, adju
  warnings.warn(
15.491676330566406
```

```
alv=var.columns[np.nonzero(alasso_result['optimal_betas'][1:])]
alv2=var[alv]


#Lasso + forward elimination

ts=time.time()
las=SelectFromModel(Lasso(alpha=.2,random_state=10))
las.fit(var,targ)
a=las.get_support()
var2=var.drop(np.array(var.columns[(las.estimator_.coef_==0).ravel().tolist()]),axis=1)

vsel=var2.iloc[:,0:1]
model=LinearRegression().fit(vsel,targ)
vselscore=(model.score(vsel,targ))
for i in np.arange(0,len(var2.columns)):
  v=var2.iloc[:,i:i+1]
  model=LinearRegression().fit(v,targ)
  vscore=model.score(v,targ)
  if(vscore>vselscore):
    vselscore=vscore
    vsel=v
var2=var2.drop(vsel,axis=1)
vog=vselscore
vt=vsel

j=0
k=0
while j==0:
  vselscore=0
  for i in np.arange(0,len(var2.columns)):
    v1=var2.iloc[:,i:i+1]
    v=pd.concat([vt,v1],axis=1)
    model=LinearRegression().fit(v,targ)
    vscore=model.score(v,targ)
    if(vscore>vselscore):
      vselscore=vscore
      vsel=v1



  if(-vog+vselscore <= .001):
    j=1

  if(-vog+vselscore>.001):
    vt=pd.concat([vt,vsel],axis=1)
    vog=vselscore
    var2=var2.drop(vsel,axis=1)
  print(vog)
  k=k+1
  if(k>40):
    j=1

print(time.time()-ts)
```

```
0.7826647641019246
0.8906682344248837
0.9031338121742389
0.909382788072085
0.9147208335562252
0.9171800828479997
0.9198402609045595
0.9220698482222343
0.9239793025917172
0.9266109722306634
```

```
0.9266109722306634
4.020595073699951
```

```
vt
```

	Reactions	Potential	Age	Special	Heading accuracy	Value (€)	Sprint speed	Composure	Strength	Ball control	Positioning
0	96.0	94	32	2228	88.0	95500000	91.0	95.0	80.0	93.0	95.0
1	95.0	93	30	2154	71.0	105000000	87.0	96.0	59.0	95.0	93.0
2	88.0	94	25	2100	62.0	123000000	90.0	92.0	53.0	95.0	90.0
3	93.0	92	30	2291	77.0	97000000	77.0	83.0	80.0	91.0	92.0
5	91.0	91	28	2143	85.0	92000000	83.0	87.0	84.0	89.0	91.0
...	...	...	...	...	...	...	...	...	...	...	...
17974	48.0	56	18	1105	43.0	40000	60.0	37.0	42.0	34.0	27.0
17975	47.0	46	28	1174	38.0	30000	22.0	42.0	37.0	46.0	39.0
17978	54.0	61	17	1194	47.0	60000	51.0	50.0	32.0	41.0	46.0
17979	49.0	64	18	1302	40.0	60000	58.0	45.0	46.0	43.0	37.0
17980	40.0	64	18	1031	48.0	50000	49.0	33.0	67.0	32.0	20.0

15352 rows × 11 columns

```
!pip install abess
from abess.linear import LinearRegression as LinReg
```

```
#perform ABESS
ts=time.time()
mod=LinReg()
mod.fit(var.to_numpy(),targ.to_numpy())
print(time.time()-ts)
```

```
1.6084558963775635
```

```
absv=var[var.columns[np.nonzero(mod.coef_)]]
```

```
#take data from R notebook to get L0Learn's dataframe
lol=np.array([1,1,1,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,1,1,0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
lol=var[var.columns[np.nonzero(lol)]]
```

```
#create 1 big array of all columns
t=np.append(lv.columns,alv)
t=np.append(t,vt.columns)
t=np.append(t,absv.columns)
t=np.append(t,lol.columns)
```

```
#figure out which ones share values in other dataframes and which is in each
np.unique(t,return_counts=True)
```

```
(array(['Acceleration', 'Age', 'Aggression', 'Agility', 'Balance',
        'Ball control', 'Composure', 'Crossing', 'Curve', 'Dribbling',
        'Finishing', 'Free kick accuracy', 'GK handling', 'GK kicking',
        'GK reflexes', 'Heading accuracy', 'Interceptions', 'Jumping',
        'LS', 'LWB', 'Long shots', 'Marking', 'Penalties', 'Positioning',
        'Potential', 'RS', 'RWB', 'Reactions', 'ST', 'Short passing',
        'Shot power', 'Sliding tackle', 'Special', 'Sprint speed',
        'Stamina', 'Standing tackle', 'Strength', 'Value (€)', 'Vision',
        'Volleys'], dtype=object),
 array([4, 5, 1, 1, 2, 5, 5, 1, 1, 1, 1, 2, 1, 2, 1, 5, 1, 1, 1, 1, 2, 3,
        1, 5, 5, 1, 1, 5, 1, 4, 1, 1, 3, 5, 4, 1, 5, 4, 3, 1]))
```

```
absv.columns
```

```
Index(['Value (€)', 'Age', 'Potential', 'Acceleration', 'Balance',
       'Ball control', 'Composure', 'Crossing', 'Finishing', 'GK kicking',
       'Heading accuracy', 'Marking', 'Positioning', 'Reactions',
       'Short passing', 'Sprint speed', 'Stamina', 'Strength', 'Vision'],
      dtype='object')
```

```
#R2 for lasso
m1=LinearRegression()
m1.fit(lv,targ)
m1.score(lv,targ)
```

```
0.929766220747539
```

```
#R2 for adaptive lasso
m2=LinearRegression()
m2.fit(alv2,targ)
m2.score(alv2,targ)
```

```
0.9258953264980461
```

```
#R2 for ABBES
m3=LinearRegression()
m3.fit(absv,targ)
m3.score(absv,targ)
```

```
0.9301295280408212
```

```
#R2 for lasso + forward elimination
m4=LinearRegression()
m4.fit(vt,targ)
m4.score(vt,targ)
```

```
0.9266109722306634
```

```
#R2 for L0Learn
m5=LinearRegression()
m5.fit(lol,targ)
m5.score(lol,targ)
```

```
0.9287436136848077
```

```
#The R code starts here
install.packages("plyr")
install.packages("Rcpp")
install.packages("reshape2")
install.packages("RcppArmadillo")
install.packages("L0Learn", repos = "http://cran.rstudio.com")


library('L0Learn')


#perform L0Learn method, get various values of lambda
st=Sys.time()
Y=read.csv('target.csv')
X=read.csv('var.csv')


Y=data.matrix(Y)
X=data.matrix(X)

print(Sys.time()-st)
```

```
Time difference of 0.5184736 secs
```

```
f<-L0Learn.fit(X,Y,penalty='L0',maxSuppSize = 20)


print(f)
```

A data.frame: 15 × 3

lambda <dbl>	gamma <dbl>	suppSize <dbl>
3.49624e-01	0	0
3.46127e-01	0	1
3.05955e-02	0	2
2.61290e-02	0	3
3.12656e-03	0	4
1.81551e-03	0	6
1.46770e-03	0	7
7.39553e-04	0	9
5.32893e-04	0	10
3.96998e-04	0	11
1.03138e-04	0	14
7.16561e-05	0	16
4.77703e-05	0	17
3.34170e-05	0	19
2.14183e-05	0	20

```
#choose lambda such that we get 14 variables returned
coef(f, lambda=0.000103138, gamma=0)
```

```
65 x 1 sparse Matrix of class "dgCMatrix"

Intercept -1.191133e+01
V1         1.136719e-07
V2         5.433629e-01
V3         4.975609e-01
V4         .
V5         3.261149e-02
V6         .
V7         .
V8         .
V9         8.672149e-02
V10        5.209585e-02
V11        .
V12        .
V13        .
V14        .
V15        .
V16        .
V17        .
V18        .
V19        .
V20        .
V21        5.311678e-02
V22        .
V23        .
V24        .
V25        .
V26        1.069562e-02
V27        .
V28       -2.235695e-02
V29        1.203743e-01
V30        3.911532e-02
V31        .
V32        .
V33        2.880280e-02
V34        2.250550e-02
V35        .
V36        3.629213e-02
V37        .
V38        .
V39        .
V40        .
V41        .
V42        .
V43        .
V44        .
V45        .
V46        .
V47        .
V48        .
V49        .
V50        .


V53        .
V54        .
V55        .
V56        .
V57        .
V58        .
V59        .
V60        .
V61        .
V62        .
V63        .
V64        .
```