Partial Least Square Regression

Yanlin Yu, Xiaoming Liu

May 1, 2019

Abstract

In this project, we explored the partial least square regression(PLS) algorithm for regression problems, compared its performances and profile with some other common regression algorithms and developed a highly optimized implementation of the algorithm. PLS is a efficient multivariate regression algorithm that is especially powerful in modelling regression problems with strongly correlated features. PLS creates intermediate latent components formed by linear combinations of the original features and use the latent components as the features for the final regression problem. In this project, we implemented the algorithm with python and optimized the performance of our implementation using Numba and Cython.

Keywords: Multivariate Regression, Partial Least Square Regression, Python, Numba, Cython

1 Background

The Partial Least Square Regression(PLS) algorithm was first purposed by Herman[1] in 1969. PLS is used to solve regression problems with multiple dependent variables, with a wide range of applications in statistical analysis.

2 Method Description

The problem we are solving can be formally defined as follows. Given N observations, with each observation a d dimensional feature vector. The observation matrix X is thus a $n * d$ matrix. Similarly, we have independent variable matrix Y, with a dimension of $N * k$, in which N is the number of observations and k is the number of independent variables for each observation.

Similar to what we do in Principle Component Analysis(PCA) in which we decompose a matrix X of rank r as a sum of matrices of rank 1, i.e, $X = \sum M_i$. The observation matrix X can be modelled by outer relation.

$$X = TP^{'} + E = \sum t_h * p_h^{'} + E \tag{1}$$

Similarly, the independent variable matrix can also be modelled by similar outer relation

$$Y = UQ^{'} + F^{*} = \sum u_h * q_h^{'} + F^{*} \tag{2}$$

Thus, the relationship between observations X and independent variables Y can be modelled

by the relationship of their individual principle components, i.e, the inner relation. The inner relation can be expressed as $u_h = b_h t_h$ where $b_n = \frac{u_h^{'} t_h}{t_h^{'} t_h}$, in which b_n is the regression coefficient for the n_{th} principle component. Let B be the regression coefficient matrix, which

$$B = \begin{bmatrix} | & | & & | \\ b_1 & b_2 & \dots & b_n \\ | & | & & | \end{bmatrix}$$

The mixed relationship can thus be represented as

$$Y = TBQ^{'} + F \tag{3}$$

with the optimization goal for our regression problem being $\min_B \|F\|$.

The detailed algorithm is listed as follows:

input : Feature Matrix X_{n*d}

Independent Variable Matrix Y

;

for *each component* **do**

1. Take u_{start} equals to a random column from Matrix Y ;
2. $w^{'} = u^{'}X/u^{'}u$;
3. $w^{'}_{new} = w^{'}_{o}ld/\left\|w^{'}_{o}ld\right\|$;
4. $t = Xw/w^{'}w$;
5. $q^{'} = t^{'}Y/t^{'}t$;
6. $q^{'}_{new} = q^{'}_{old}//normq^{'}_{old}$;
7. $u = Yq/qq^{'}$;
8. **if** *the new t is close to t from the previous iteration* **then**

   go to step 9 ;

   else

   go to step 2;

   end
9. $p^{'} = t^{'}X/t^{'}t$;
10. $p^{'}_{new} = p^{'}_{old}/\left\|p^{'}_{old}\right\|$;
11. $t_{new} = t_{old}\left\|p^{'}_{old}\right\|$;
12. $w^{'}_{n}ew = w^{'}_{o}ld\left\|p^{'}_{old}\right\|$;
13. The Regression Coefficient b is thus: $b = u^{'}t/t^{'}t$;

end

;

Algorithm 1: PLS Algorithm

3 Implementation and Optimization

In our project, we implemented partial least square algorithms in following ways and compared on their performances and profiles:

1. Pure Python implementation
2. Numba
3. Cython

3.1 Numba

Numba is a Just-in-Time compiler that accelerates Python and NumPy code by translating Python functions to optimized machine code at runtime.

3.2 Cython

Cython is a programming language that produces C-Style extensions for Python codes. While Python is a dynamically typed interpreted language, Cython is a compiled programming language that introduces static typing to enhance the runtime efficiency of the code. Moreover, as Cython functions will be first interpreted into C code when compiling, it can obtain similar runtime efficiency to C functions, which performs far more efficient than Python codes.

To optimize the running profile using Cython, we managed to replace some of the pure python Numpy functions, which create performance bottlenecks for our implementation, with typed Cython functions. For instance, by rewriting *numpy.linalg.norm* to Cython version which is

optimized specifically for 1-D vectors, the efficiency of the function increased by over 20 times.

4 Comparative Analysis with Competing Algorithms

To help profile the performance of our PLS algorithm, we decide to compare the result of our model with that of using a Ordinary Least Squares(OLS) method and a Gaussian Process Regression(GPR).

4.1 OLS

Ordinary Least Squares regression (OLS) is more commonly named linear regression. The linear regression model stated in terms of a well-defined population is:

$$\begin{aligned} y &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k + u \\ &= \beta_0 + x\beta + u \end{aligned} \tag{4}$$

where x is $n \times 1$ and observed, β is $k \times 1$ vector of unknown "slope" parameters.

OLS algorithm corresponds to minimizing the sum of square differences between the observed and predicted values.

There are two assumptions for OLS.

1. Assumption OLS.1 (Zero Correlation)

   The error term has a population mean of zero and uncorrelated with each explanatory variables.

$$E(x'u) = 0 \tag{5}$$

2. Assumption OLS.2 (No Perfect Collinearity)

   In the population, there are no independent variable has a perfect linear relationship with other covariates.

$$rankE(x'x) = k \tag{6}$$

Under OLS.1 and OLS.2, β is identified, which can be written as a function of population moments in observable variables.

$$\begin{aligned} x'y &= x'(x\beta) + X'u \\ E(x'y) &= E(x'x)\beta + E(x'u) \\ E(x'y) &= E(x'x)\beta, \text{ by Assumption OLS.1} \\ \beta &= [E(x'x)]^{-1}E(x'y), \text{ by Assumption OLS.2} \end{aligned} \tag{7}$$

Then, replace the population means with sample means we can get:

$$\begin{aligned} \hat{\beta} &= (N^{-1}\sum_{i=1}^{n} x_i'x_i)^{-1}(N^{-1}\sum_{i=1}^{n} x_i'y_i) \\ &= (X'X)^{-1}(X'Y) \end{aligned} \tag{8}$$

where X is $N \times k$ and Y is $N \times 1$. This is how OLS works to estimate parameters.

4.2 GPR

Gaussian Process Regression is a very powerful non-linear prediction tool. Imagine we observe a dataset $D = \{(X_i, y_i)_{i=1}^n\} = (X, y)$ from the model:

$$Y_i = f(x_i) + \epsilon_i, \epsilon_i \overset{i.i.d}{\sim} N(0, \sigma^2) \tag{9}$$

We use a Gaussian process to place a prior on the space of functions.

$$f(x_i) \sim GP(.|0, K) \tag{10}$$

The objective is to specify the predictive distribution on Y^* which will be multivariate normal.

$$Y^*|X^*, X \sim N(\mu^*, \Sigma^*) \tag{11}$$

Given that prior on f is a GP, likelihood is Gaussian, therefore posterior on f is also a GP. We can use this to make predictions.

$$p(Y^*|X^*, D) = \int p(Y^*|X^*, f, D)p(f|D)df \tag{12}$$

4.3 Comparison on Simulated Dataset

4.3.1 Synthesized Dataset with Single Regression Target

The runtime and performance statistics is obtained using models trainined on synthetic dataset with 19 linearly dependent features and 5000 data points for training. The MSE is obtained by testing the model performance on a generated test set of size 100.

Below are performance for OLS, GPR, the original PLS algorithm and a optimized PLS method. We only use 1-D PLS algorithm to provide a simple profile for the performance of our model compared to OLS and GPR here. More about n-D PLS performance will be discussed later.

Algorithms	Runtime	MSE
Ordinary Least Squares(OLS)	1.97 ms ± 25.2 s per loop	0.076
Gaussian Process Regression(GPR)	13.2 s ± 329 ms per loop	0.076
Original PLS1	1.57 ms ± 29.7 s per loop	0.077
Optimized PLS1 using Cython	1.13 ms ± 60.6 s per loop	0.076

Table 1: Performance for Competing Algorithms

As we can see from Table 1, the MSE of all four methods are almost the same. However, the efficiency of them differs from each other. GPR is much slowest compared to other algorithms, taking around 10000 times the runtime of optimized PLS1. The efficiency of OLS is higher than GPR, but still relatively below PLS1 algorithms. Also, after optimizing the original PLS1 algorithm with Cython, it achieves a even better MSE with a lower runtime. Thus, our implementation using Cython has optimized the original implementation significantly.

4.3.2 Synthesized Dataset with Multiple Regression Target

The dataset is synthesized similarly with 3 linearly independent features and with 16 features linearly dependent on the independent features. The dataset contains 3 regression target, each a different linear combination of the features. The dataset consists of 5000 data points for training and 100 data points for testing.

Algorithms	Runtime	MSE
Gaussian Process Regression(GPR)	13.2 s ± 329 ms per loop	0.076
Original PLS	2.7 ms ± 13.8 s per loop	0.077
Optimized PLS using Cython	1.83 ms ± 9.65 s per loop	0.076
Optimized PLS using Numba	2.47 ms ± 20.1 s per loop	0.076

Table 2: Performance for Competing Algorithms

5 Application to Real Datasets

5.1 Red Wine Quality Dataset

5.1.1 Performance with Single Regression Target

This datasets is related to red variants of the Portuguese "Vinho Verde" wine, which recorded the physicochemical properties of different wines and their quality grade[2]. The dataset consists of 1599 data samples, and each data point in the dataset consists of 11 features:

1. fixed acidity
2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides
6. free sulfur dioxide
7. total sulfur dioxide

8. density
9. pH
10. sulphates
11. alcohol

The label of the dataset is the final quality grade of each wine, from a score of 0 to 10. The performance and runtime profile for each algorithms are listed in the table below.

Algorithms	Runtime	MSE
Ordinary Least Squares(OLS)	4.35 ms ± 176 s per loop	0.3837
Gaussian Process Regression(GPR)	6.73 s ± 87 ms per loop	0.3859
Original PLS1	2.62 ms ± 114 s per loop	0.4007
Optimized PLS1	2.24 ms ± 119 s per loop	0.3842

Table 3: Performance on Red Wine Quality Dataset

5.1.2 Performance with Multiple Regression Target

In order to make the dataset applicable in multiple regression target setting, we extracted the feature 'pH Value' as the second regression target and run our multivariate PLS on the new dataset. The detailed performance is illustrated below.

Algorithms	Runtime for PLS	MSE
Original PLS	7.55 ms ± 165 s per loop	1.3201
Optimized PLS using Cython	5.03 ms ± 79.6 s per loop	1.3201
Optimized PLS using JIT	5.03 ms ± 270 s per loop	1.3201

Table 4: Performance for multivariate PLS on Red Wine Quality Dataset

6 Conclusion

By making comparison in terms of runtime efficiency and accuracy with similar regression algorithms, we can conclude that PLS is one of the better methods to apply in such settings, especially in regression problems with multiple regression targets.

Moreover, our project further illustrated how Numba and Cython based optimization can effective boost the runtime efficiency of Python programming with Numpy in scientific computation settings.

References

[1] W. Herman, "Estimation of principal components and related models by iterative least squares," *Journal of Multivariate Analysis*, Jan 1969.

[2] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decis. Support Syst.*, vol. 47, pp. 547–553, Nov. 2009.