Ordinary least squares regression of Franke's function

Anders Eriksen

Contents

List of Figures

List of Tables

Abstract

The main summary of the work

1 introduction

aims and rationale of the physics case, what you've done as well as a brief summary of the report
o Motivate the reader
o What done
o Structure of report

2 Methods

Theoretical models and technicalities.
o describe methods and algorithms
o explain implementation of methods and say something about the structure of algorithm and present parts of code
o Plug in some calculations to test code, such as selected runs used to validate and verify results. Latter extremely important. Reader needs to know that your code reproduces selected benchmarks and reproduces previous results, either numerical and/or well-known closed-form expressions.

The aim is to study regression models on data. These create a continuous function with which the input data is fitted. The first, and most basic is the *Ordinary Least Squares method* (*OLS*)

To test and validate the algorithms, a closed form function is an advantage. We wish to predict terrain. there is a 2-dimensional function called *Franke's function*[] which can transform a set of coordinate vectors with values between $[0, 1]$ into height values through a sum of weighted exponents.

$$\begin{aligned} f(x,y) \ &= \frac{3}{4}\exp\Big\{-\frac{(9x\ -\ 2)^2}{4}\ -\ \frac{(9y\ -\ 2)^2}{4}\Big\} \\ &+\frac{3}{4}\exp\Big\{-\frac{(9x\ +\ 1)^2}{49}\ -\ \frac{(9y\ +\ 1)}{10}\Big\} \\ &+\frac{1}{2}\exp\Big\{-\frac{(9x\ -\ 7)^2}{4}\ -\ \frac{(9y\ -\ 3)^2}{4}\Big\} \\ &-\frac{1}{5}\exp\big\{-(9x\ -\ 4)^2\ -\ (9y\ -\ 7)^2\big\} \end{aligned} \tag{1}$$

To generate the data, one can create a uniformly distributed set of initial values ordered from low to high. These initial arrays are then combined into coordinate data through the numpy meshgrid function before being passed to Franke's function. These are matrices, and they are therefore flattened to 1D arrays through numpy's *ravel()* function.

Next comes setting up the model. Firstly, constructing the model design matrix. We want a design matrix for variable levels of model complexity, so we construct it from a polynomial combination of the input parameters x and y.

Next is to fit the model. The linear model, taken from Hastie et al[], predicts the value Y as

$$Y\ =\ X^T\,\beta. \tag{2}$$

Algorithm 1 make design matrix X given input $\vec{x}, \vec{y}$ and dimension n

create X as a matrix of ones with dimension $length(x)$ and $(int)i(i+1)/2$
for $i = 1$ **to** $n + 1$ **do**
  $q = (int)i(i+1)/2$
  for $k = 0$ **to** $i + 1$ **do**
    $X[:, q + k] = x^{i-k} \cdot y^k$
  end for
end for

Y is given through the inner product of the transpose of X and β. X being the design matrix mentioned above.

One method to approximate this, is with the *Residual Sum of Squares*

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2.[] \tag{3}$$

As the sum of squares, there is a guaranteed minimum, though not necessarily a unique one. If we write this in vector notation, differentiate w.r.t β we get the so-called *normal equations*.

$$\begin{aligned} &RSS(\beta) = (\vec{y} - \mathbf{X}\beta)^T (\vec{y} - \mathbf{X}\beta) \\ &\text{differentiating w.r.t } \beta \text{ gives:} \\ &\mathbf{X}^T(\vec{y} - \mathbf{X}\beta) = 0 \\ &\text{given non-singular } \mathbf{X}^T\mathbf{X}\text{:} \\ &\hat{\beta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T \vec{y}. \end{aligned} \tag{4}$$

A prediction can then be made of the values Y given our β and $\mathbf{X}$ with

$$\hat{Y} = \mathbf{X}\,\hat{\beta}. \tag{5}$$

In a naive way, this prediction can be compared to the reference output values, to see how well the model predicts the data. Our wish, however, is to predict data of the same stochastic "source" as our known data. As such, we could get arbitrarily like the known data by increasing the complexity of the polynomial we use to fit the input. This will, however cost us generality. Any data outside the data we use to specifically fit the model is unlikely to fall within the model's prediction. This bias is something we need to avoid.

To this effect, we can implement a sectioning of the available data so that we can use parts of the set to "train" our model and a separate part as a

"test" or "validation". The meethod chosen is called *k-fold Cross Validation* (k-fold CV). The data set is split into k different "folds". Then, on a rotation, 1 fold is designated as the "test" fold and the remaining folds are designated "training". The model if fitted to the training set and then tested on the validation set. One can then plot the MSE for the training and test folds side by side, over different degrees of model complexity. What one should expect to see here, is at one point the errors diverge between the training and test sets due to over-fitting. As mentioned above, the model predicts closer and closer the set used to train, but sets that diverge from the training are less and less in accordnance with the model.

Algorithm 2 k fold split based on input data $\vec{x}$

```
make index array (inds) with elements from 0, to the length of x
randomize inds and partition into k folds of size length(x)/k,
gather into array of arrays k, length(x)/k of indices, folds
for fold in; folds do
    test = input[fold]
    train = input[!fold]
    find β as in eq(4) using train
    predict output using β
    compare predict and test as well as predict and train
end for
```

While this method should produce great results for our known function, under normal circumstances the distribution function is either unknown or nonexistent. An appropriate example here could be measured terrain data. There is no overarching function decinding the height of an area. As such, and especially so for large data sets, There is a need to control the outcommes of our minized betas. Not to mention our minimization assumes the matrix $\mathbf{X}^T\mathbf{X}$ is non-singular, or in other words is invertible. We need to control for abberrant β's so as to curb some of the variance we could see on large datasets from cancelling pairs of positive and negative coefficients. Shringing such "wild" coefficients would allow for a reduction in variance without much gain in bias. due to shrinking these coefficients, the following methods are also called *Shrinkage Methods*.

One such, is Ridge regression. Here, we add a parameter λ, so that

$$\hat{\beta}^{ridge} = argmin\left\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}(x_{ij}\beta_j))^2 + \lambda\sum_{j=1}^{p}\beta_j^2\right\} \tag{6}$$

$\lambda \geq 0.$

A large λ means that $\hat{\beta}$ experiences a greater shrinkage in order to fit. A point also, is that β_0 is not penalized like this. If we were to center the results, so that we do not fit for x_{ij}, but for $x_{ij} - \overline{x}_j$ and β_0 by $\overline{y} = \frac{1}{N}\sum_1^N y_i$. Centering the data like this allows the rewrite of (6) to matrix form with the residual sum of squares

$$RSS(\lambda) = (\vec{y} - \mathbf{X}\beta)^T(\vec{y} - \mathbf{X}\beta) + \lambda\beta^T\beta, \tag{7}$$

with a ridge regression solution of:

$$\beta^{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{1})^{-1}\mathbf{X}^T\vec{y} \tag{8}$$

And with the added diagonal, we essentially avoid a singular $\mathbf{X}^T\mathbf{X}$.

Should the matrix $\mathbf{X}^T\mathbf{X}$ be singular there is still a way to invert it, using singular value decomposition (SVD). This method expresses a matrix as an inner product of 3 different matrices[]

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T. \tag{9}$$

here, if $\mathbf{X}$ is a $n \times p$ matrix, then $\mathbf{U}$ is an $n \times n$ matrix, $\mathbf{D}$ is an $n \times p$ matrix, and $\mathbf{V}$ is a $p \times p$ matrix. $\mathbf{D}$ is a diagonal matrix, whose diagonal values are called the singular values of $\mathbf{X}$. If any of $d_{i,i} = 0$ then $\mathbf{X}$ is singular.

With some simplification, it can be shown,

$$\begin{aligned}\mathbf{X}\beta &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y} \\ &= \dots \\ &= \mathbf{U}\mathbf{U}^T\vec{y}\end{aligned} \tag{10}$$

Which we can send into the ridge solutions (8) so

$$\begin{aligned}\mathbf{X}\hat{\beta}^{ridge} &= \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{1})^{-1}\mathbf{X}^T\vec{y} \\ &= \mathbf{U}\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{1})^{-1}\mathbf{D}\mathbf{U}^T\vec{y} \\ &= \sum_{j=1}^{p}\vec{u}_j\frac{d_j^2}{d_j^2+\lambda}\vec{u}_j^T\vec{y}.\end{aligned} \tag{11}$$

Here, $\vec{u}_j$ are the collumn vectors of $\mathbf{U}$, and d_j are the singular values of $\mathbf{X}$, the diagonal elements of $\mathbf{D}$. As the parameter $\lambda \geq 0$, the fraction $\frac{d_j^2}{d_j^2+\lambda} \leq 1$, and the β's will be shrunk in accordance with the size of the singular values. The collumns of $\mathbf{V}$ from (9) form the so-called *principal component directions* of $\mathbf{X}$. The first of these, has the property where $z_1 = \mathbf{X}v_1$ has the larges sample variance. amongst all normalized collumns of $\mathbf{x}$

$$Var(\vec{z}_1) = Var(\mathbf{X}\vec{v}_1) = \frac{d_1^2}{N}, \tag{12}$$

and further, $\vec{z}_1 = \vec{u}_1\, d_1$. $\vec{z}_1$ is the first principal component of $\mathbf{X}$, making $\vec{u}_1$ the *normalized* first principal component. subsequent principal components have similar variance $\vec{z}_j = \frac{d_j^2}{N}$ and are orthogonal to the previous. The last principal component sports the lowest variance.

$\mathbf{X}$ projects the output Y with greater variance along the prior principal component directions rather than the latter. A surface plot along these projections will then have less variance in the gradient along the axes of greater projected variance. Ridge regression shrinks the lower variance and screens the output from the gradient variance.

The assumption behind this is that the response should vary most along the direction of greatest variance of the input. This is not in general true, but inputs are often chosen to study because the output vary along with them.

Another way to shrink the coefficients in *beta* is with minimizing with a restriction

$$\hat{\beta}^{lasso} = argmin \left\{ \frac{1}{2} \sum_{i=1}^{N} \left(y_i \, - \, \beta_0 \, - \, \sum_{j=1}^{p} x_{ij} \beta_j \right)^2 \, + \, \lambda \sum_{j=1}^{p} |\beta| \right\}. \qquad (13)$$

Where the main difference here is the sum of $|\beta|$ rather than β^2 as in ridge (6). The main difference here, is that there is now a non-linear solution with y_i. This also means that there is no closed-form solution, and the computation is therefore far more taxing. There are libraries which solve this at roughly the same computational cost as ridge, however. In this example, the library used is python's scikitlearn. []

3 Results

The results and discussion of such
o Present results
o Give critical discussion of you work & place it in correct context
o Relate work to other calculations/studies
o Reader should be able to reproduce calculations should they wish to do so. All input variables should be properly explained.
o Make sure figures and tables contain enough information in their captions. Axis labels, etc. A reader should be able to get a first impression of the work by purely studying the figures and tables.

4 Conclusion

Conclusions and perspectives
o State main findings and interpretations
o Try as far as possible to present perspectives for future work.
o Try to discuss the pros and cons of the methods and possible improvements.

5 Appendix

any extra material
o Additional calculations used to validate code.
o Selected calculations. Can be listed with few comments.
o Listing of code if necessary.
Consider moving parts from methods to appendix. A webpage is also an appropriate place for a lot of this type of info.

o Always reference material you base your work on, either scientific articles/reports or books.
o Refer to articles as: name(s) of author(s), journal, volume(Bold), page and year in parenthesis.
o Refer to bookds as: name(s) of author(s), title of book, publisher, place and year, eventual page numbers.