Investigating the mathematics behind statistical modelling

Motivation

A statistical model is a mathematical model built upon a set of statistical assumptions regarding the generation of a data set. Linear models generalised linear models (GLMs) and generalised additive models (GAMs) are common statistical models used to investigate potential relationships between covariates in a dataset. Constructed in R or Python, an understanding of the fundamental mathematics holding the statistical model is often not required as various libraries, methods and classes have been written to account for this. Thus, the main pre-requisites for the user includes a basic understanding of various statistical models, which methods to use, how to choose and compare models and how to understand the diagnostics and properly interpret statistical indicators before refining the final model.

The aim of this project was to understand the mathematics and statistics used to build the linear model and then to construct it in Python. Many methods have already been implemented in libraries such as finding the prediction coefficients to minimise the residual errors, r-squared and adjusted r-squared values, mean squared error, residual standard errors and prediction errors for coefficients. These will all be coded from scratch with minimal use of existing Python libraries. The end result of the project will have a statistical model on a data set which returns all the required statistical indicators and also allows the user to modify the model for deeper analysis.

Libraries

The libraries used in this project includes:

os	used to get the base directory containing the datasets
pandas	used to read the dataset into a pandas data frame
numpy	used for linear algebra calculations and storage of matrices

Methodology

In the linear model, a response variable is modelled with a variety of covariates using a linear function (See Appendix A). Using the observed values, values of the $\hat{ß}$ values for each covariate and the intercept (if included in the model) can be estimated through least squares (See Appendix B). An estimation of the variance in the model will be obtained and the standard errors for the $\hat{ß}$ values will be obtained through calculation of the covariance matrix (See Appendix C and D). Various statistics of the model such as r-squared, adjusted r-squared, residual sum of squares etc will also be evaluated (See Appendix E).

The model was built in an object-oriented approach in which linear model objects can be created from the linear model class. Each linear model object has the following attributes:

dataset	string	name of the dataset
dependent	list	response variable
independent	list	covariates
intercept	boolean	should an intercept be used

Once a linear model object has been created, all the relevant statistics and estimates can be obtained. The information below describe the purpose of each function in the class. Further details on function implementation can be found in the respective appendix sections.

Three functions were used to set up and return all the required data for the model:

get_data	reads in the dataset and returns a pandas data frame
get_response	returns the response variable data
get_design_matrix	returns the design matrix

Using least squares estimation through QR decomposition by the Gram-Schmidt process, $\widehat{ß}$ estimates can be obtained (See Appendix B). The following functions are used:

find_u	method to find matrix u (matrix with new column vectors)
find_q	method to find matrix q (orthogonal matrix)
find_r	method to find matrix r (upper triangular matrix)

The following methods are used to evaluate the model variance estimate, $\widehat{ß}$ estimates (See Appendix B) and he $\widehat{ß}$ standard errors (See Appendix D):

get_var	method to estimate the model variance
get_beta	method to find the $\widehat{ß}$ estimates
get_beta_errors	method to find the $\widehat{ß}$ standard errors

The following methods are used to return various statistics of the model (See Appendix E):

get_r_squared	method that returns the r-squared
get_adj_r_squared	method that returns the adjusted r-squared
get_rss	method that returns the residual sum of squares
get_mse	method that returns the mean squared error
get_rse	method that returns the residual standard error

The following are methods used for matrix manipulation:

dot_p	method that evaluates the dot product between two vectors
proj_u_a	method that evaluates the projection of vector a on vector u
find_col_mag	method that evaluates the magnitude of a column in a matrix
place_column	method that places the values of a list into a column of a matrix

Limitations

The model is able to model one response variable against various continuous covariates. However, the use of factor variables as covariates was not implemented and the model will break of used. Also, transformations of variables and non-linear modelling have also not been accounted for. Diagnostic plots such as QQ-plot and residuals vs fitted are also not implemented and normality should be assumed when using the model. The model is built solely for learning purposes and should not be the preferred option for practical cases as the linear model libraries in Python and R are able to produce the same results and also accounts for all of the above limitations.

Improvements

All the limitations discussed above can be ideas for improvement and if implemented the model has potential for practical use. Also, other linear algebra methods such as evaluating the inverse, transpose and matrix multiplication can be coded from scratch to eliminate the need of the *numpy* library. In addition, using a list of lists to store matrices, finding a way to read in the data without *pandas* and avoiding *os* to return the base directory will omit the use of the remaining libraries. Finally, more efficient algorithms for all the functions implemented or coding in a low-level language such as C or Fortran will also increase the calculation speed of the model.

If the project is to be repeated, an object-oriented approach would still be preferred and most of the functions can be reused. An external matrix class can be written from scratch to contain all of the matrix algebra methods, and matrix objects can be created from this class when required. Methods for F-statistics, p-value, AIC and residuals can also be implemented for deeper analysis.

Appendix

Appendix A (Linear Model Theory)

For the regression case, the statistical model can be formulated as:

$$\boldsymbol{Y_i} = ß_0 + ß_1 \boldsymbol{x_1} + ß_2 \boldsymbol{x_2} + \cdots + ß_n \boldsymbol{x_n} + \boldsymbol{\epsilon_n}$$

where:

$\boldsymbol{Y_i}$	response variable
$ß_n$	coefficient terms
$\boldsymbol{\epsilon}$	random variables representing errors

Estimation of the $ß_n$ coefficient terms can be achieved through least squares estimation where the residual sum of squares $\sum_i (\boldsymbol{y_i} - \boldsymbol{\hat{y}_i})^2$ is minimised. Another important assumption is that the $\boldsymbol{\epsilon_n}$ random variables are normally distributed.

Appendix B (Least squares estimation through QR decomposition)

The beta estimates was evaluated in the function *get_beta* through QR decomposition by the Gram-Schmidt Process using the following formula:

$$\widehat{ß} = \boldsymbol{R^{-1} Q^T y}$$

where:

$\widehat{ß}$	beta estimates
$\boldsymbol{R^{-1}}$	inverse of matrix r
$\boldsymbol{Q^T}$	transpose of matrix q
$\boldsymbol{y}$	response variable data

The Gram-Schmidt process is as follows:

Requirements:

A matrix $\boldsymbol{X}$ with columns $[\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \ldots \boldsymbol{x_n}]$

$\boldsymbol{proj_u x} = \frac{<u,x>}{<u,u>}\boldsymbol{u}$ where $< \boldsymbol{u}, \boldsymbol{x} >$ is the dot product between vectors $\boldsymbol{u}$ and $\boldsymbol{x}$

$||\boldsymbol{u}||$ = magnitude of vector u

Process:

$$\boldsymbol{u_1} = \boldsymbol{x_1} \qquad \boldsymbol{e_1} = \frac{\boldsymbol{u_1}}{||\boldsymbol{u_1}||}$$

$$\boldsymbol{u_2} = \boldsymbol{x_2} - \boldsymbol{proj_{u_1} x_2} \qquad \boldsymbol{e_2} = \frac{\boldsymbol{u_2}}{||\boldsymbol{u_2}||}$$

$$\boldsymbol{u_3} = \boldsymbol{x_3} - \boldsymbol{proj_{u_1} x_3} - \boldsymbol{proj_{u_2} x_3} \qquad \boldsymbol{e_3} = \frac{\boldsymbol{u_3}}{||\boldsymbol{u_3}||}$$

.

.

.

$$\boldsymbol{u_k} = \boldsymbol{x_k} - \sum_{j=1}^{j=k-1} \boldsymbol{proj_{u_j} x_k} \qquad \boldsymbol{e_k} = \frac{\boldsymbol{u_k}}{||\boldsymbol{u_k}||}$$

Matrix $\boldsymbol{Q}$ is defined as $[\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}, \ldots \boldsymbol{e_n}]$

Matrix $\boldsymbol{R}$ can be calculated by $\boldsymbol{Q^T X}$

Important Note: $\boldsymbol{X} = \boldsymbol{QR}$

Appendix C (Estimation of model variance)

The variance of the model $\boldsymbol{\sigma^2}$ was estimated in the function *get_var* using the formula:

$$\boldsymbol{\hat{\sigma}^2} = \frac{\sum_i (\boldsymbol{y_i} - \boldsymbol{\hat{y}_i})^2}{\boldsymbol{n} - \boldsymbol{p}}$$

where:

$\boldsymbol{\hat{\sigma}^2}$	estimated variance of $\boldsymbol{\sigma^2}$
$\boldsymbol{y_i}$	$\boldsymbol{i^{th}}$ response value
$\boldsymbol{\hat{y}_i}$	$\boldsymbol{i^{th}}$ predicted response value
$\boldsymbol{n}$	number of observations in dataset
$\boldsymbol{p}$	number of covariates in dataset

Appendix D (Calculation of $\widehat{ß}$ standard errors)

The covariance matrix of $\widehat{ß}$ was calculated in the function *get_beta_errors* using the formula:

$$\widehat{ß} = \boldsymbol{R^{-1}R^{-T}\sigma^2}$$

where:

$\widehat{ß}$	covariance matrix of $\widehat{ß}$
$\boldsymbol{R^{-1}}$	inverse of matrix r
$\boldsymbol{R^{-T}}$	transpose of the inversed matrix r
$\boldsymbol{\sigma^2}$	variance of the model

Appendix E (Statistics of the linear model)

R-Squared

$$\boldsymbol{r_{squared} = 1 - \frac{Residual\ Sum\ of\ Squares}{Total\ Sum\ of\ Squares}}$$

if an intercept is used in the model:

$$\boldsymbol{r_{squared} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}}$$

where: $\boldsymbol{\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}}$ where $\boldsymbol{n}$ is the number of observations

if no intercept is used in the model:

$$\boldsymbol{r_{squared} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i {y_i}^2}}$$

Adjusted R-Squared

if an intercept is used in the model:

$$\boldsymbol{r_{adjusted\ squared} = 1 - \frac{(1 - r_{squared})(n-1)}{n-k}}$$

if no intercept is used in the model:

$$\boldsymbol{r_{adjusted\ squared} = 1 - \frac{(1 - r_{squared})(n-1)}{n-k-1}}$$

where:

$\boldsymbol{n}$ = number of observations
$\boldsymbol{k}$ = number of covariates

Residual Sum of Squares

Residual Sum of Squares (RSS) is the sum of the squared difference between each respective predicted value and observed value.

$$\boldsymbol{RSS = \sum_i (y_i - \hat{y}_i)^2}$$

Mean Squared Error

Mean Squared Error (MSE):

$$\boldsymbol{MSE = \frac{\sum_i (y_i - \hat{y}_i)^2}{n}}$$

where n is the number of observations.

Residual Standard Error

Residual Standard Error (RSE):

$$\boldsymbol{RSE = \frac{\sum_i (y_i - \hat{y}_i)^2}{n - k + 1}}$$

where:

$\boldsymbol{n}$ is the number of observations
$\boldsymbol{k}$ is the number of covariates

Appendix F (Execution)

Execution through Terminal

1) Open the LinearModel.py file
2) Create a LinearModel object at the bottom of the file
    a. *lm1 = LinearModel("dataset", [response column], [covariates], Intercept)*
    b. Call any statistics to be returned
        i. *print(lm1.get_beta())* → gets the beta value of the model lm1
3) Open terminal and enter directory of file
4) Run the python3 file using *python3 LinearModel.py*
5) All of the statistics will be printed in terminal