Math 533 – Coding Project

Li, Xiao Qi

Moisescu, Mark-Anthony

Department of Mathematics & Statistics

McGill University

Montréal, Québec, Canada

December 17, 2023

Contents

1 Introduction

This Regression package includes classes which allow easy implementation of OLS, WLS, and Ridge estimator linear models and an ANOVA model. For all tests and confidence intervals of the OLS model, we are assuming a Gaussian linear model.

To use this package in your own project, simply put the *regression* folder in the project directory and import the required classes and methods as shown in each section below.

For a demonstration of how to use the package, look at the *demo.ipynb* notebook.

2 OLS Model

```
# Import the model
from regression.estimators import OLS
```

The OLS estimator (of β) is the solution to the following optimization problem:

$$\hat{\boldsymbol{\beta}}_{\text{ols}} \in \arg\min_{u \in \mathbb{R}^p} \frac{1}{n} \sum_{i \in [n]} (y_i - \mathbf{x}_i^\top \mathbf{u})^2$$

Furthermore, if the design matrix is full-rank then we have that the OLS estimator is unique.

2.1 Initializing the model

```
model = OLS(standardize, add_bias)
```

Parameters:

- standardize: *bool, default = True*
  - Whether to standardize the data.
- add_bias: *bool, default = True*
  - Whether to add an intercept to the data.

Attributes:

- standardize_flag: *bool*
  - Determined by the *standardize* parameter.
- add_bias_flag: *bool*
  - Determined by the *add_bias* parameter.
- beta: *array of shape (p, 1)*
  - Contains the OLS estimator coefficients ($\hat{\boldsymbol{\beta}}_{\text{ols}}$).
- yh: *array of shape (n, 1)*
  - Estimated response values ($\hat{\mathbf{y}}$).
- residuals: *array of shape (n, 1)*
  - Difference between actual $\mathbf{y}$ values and estimated $\hat{\mathbf{y}}$ values ($\hat{\mathbf{e}}$).
- sigma2_naive: *float*
  - Mean of the squared residuals ($\hat{\sigma}^2_{naive}$).
- sigma2_corrected: *float*
  - Unbiased version of $\hat{\sigma}^2_{naive}$ ($\hat{\sigma}^2_{cor}$).

2.2 Methods

2.2.1 Calculating the OLS estimator

```
model.fit(x, y)
```

Calculates the following:

- $\hat{\boldsymbol{\beta}}_{\text{ols}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$
- $\hat{\mathbf{e}} = \mathbf{y} - \hat{\mathbf{y}}$
- $\hat{\sigma}^2_{naive} = \frac{1}{n} \sum_{i=1}^{n} \hat{e}_i$
- $\hat{\sigma}^2_{cor} = \frac{n}{n-p} \hat{\sigma}^2_{naive}$

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- y: *array of shape (n,1)*
  - Vector of response variables.

2.2.2 Estimating and predicting the response

```
model.predict(x)
```

Returns $\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}_{\text{ols}}$.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.

2.2.3 Calculating the R^2 score

```
model.score(y, adjusted = False)
```

Calculates and returns the $\mathbf{R}^2$ score:

$$\mathbf{R}^2 = \frac{\sum_{i \in [n]} (\hat{y}_i - \bar{y})^2}{\sum_{i \in [n]} (y_i - \bar{y})^2}$$

or the adjusted $\mathbf{R}^2$:

$$\mathbf{R}^2_{\text{adj}} = 1 - (1 - \mathbf{R}^2) \cdot \frac{n}{n - p}$$

Parameters:

- y: *array of shape (n, 1)*
  - Vector of true response values.
- adjusted: *bool, default = False*
  - Whether to calculate the adjusted R^2 score.

2.2.4 Calculating the MSE

```
model.mse(y)
```

Calculates and returns the MSE:

$$\mathbf{MSE}(\hat{\boldsymbol{\beta}}_{\text{ols}}) = \frac{1}{n} \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$$

Parameters:

- y: *array of shape (n, 1)*
  - Vector of true response values.

2.2.5 Hat Matrix

For OLS, we define the Hat Matrix as the following:

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top$$

```
ols_model.hat_matrix(x)
```

Returns the Hat Matrix under OLS.

2.2.6 Tests

Testing individual coefficients using T-test

```
model.t_test(j, bj, alpha, use_p_value = False, print_output = True)
```

Returns TRUE if $H_0 : \hat{\beta}_j = b_j$ is accepted, FALSE if rejected.

Parameters:

- j: *int*
  - Index of the coefficient to test.
- bj: *int*
  - Coefficient value to test.
- alpha: *float*

  - Significance level of the test.
- use_p_value: *bool, default = False*
  - Whether to test using p-value or critical value.
- print_output: *bool, default = True*
  - Whether to print out the test decision.

Testing a reduced model using an F-test

```
model.f_test_reduced_model(x_reduced, y, alpha, print_output = True)
```

Returns TRUE if H_0: "The reduced model M_1 is correct" is accepted, and FALSE if rejected.

Parameters:

- x_reduced: *array of shape (n, p1)*
  - Reduced matrix of covariates to test.
- y: *array of shape (n, 1)*
  - Vector of response values.
- alpha: *float*
  - Significance level of the test.
- print_output: *bool, default = True*
  - Whether to print out the test decision.

Testing arbitrary number of constraints using an F-test

```
model.f_test_constraints(R, r, alpha, print_output = True)
```

Test for k constraints simultaneously. Note: Do not forget to include a first column constraint for the intercept. Returns TRUE if H_0 : $\mathbf{R}\boldsymbol{\beta} = \mathbf{r}$ is accepted, and FALSE if rejected.

Parameters:

- R: *array of shape (k, p)*
  - Matrix of constraints. Each row represents a constraint.
- r: *array of shape (k, 1)*

  - Value for each constraint.
- alpha: *float*
  - Significance level of the test.
- print_output: *bool, default = True*
  - Whether to print out the test decision.

2.2.7 Confidence intervals

Confidence interval for a single coefficient

```
model.confidence_interval_single(j, alpha, print_output = True)
```

Returns (left, right) values of the $(1 - \alpha)\%$ confidence intervals for β_j.

Parameters:

- j: *int*
  - Index of coefficient for which to compute the confidence interval.
- alpha: *float*
  - Significance level of the confidence interval.
- print_output: *bool, default = True*
  - Whether to print out the confidence interval description.

Bonferonni confidence intervals

```
model.confidence_interval_bonferonni(coefficients, alpha,
                                     print_output = True)
```

Returns a list of tuples (where each tuple represents a confidence interval for a specific coefficient).

Parameters:

- coefficients: *array of shape (1,p)*
  - List of integers that represent the indices of the coefficients for which confidence intervals are to be computed.
- alpha: *float*
  - Significance level of the test.
- print_output: *bool, default = True*
  - Whether to print out the confidence region description.

Ellipsoid confidence intervals

```
model.confidence_ellipsoid_test(test_beta, alpha, print_output = True)
```

Returns TRUE if the specified 'test_beta' falls within the confidence region for the regression coefficients, and FALSE otherwise.

Parameters:

- test_beta: *array of shape (1, p)*
  - A point in the parameter space for which the test is performed to check whether it lies within the confidence ellipsoid.

- alpha: *float*
  - Significance level of the confidence region.
- print_output: *bool, default = True*
  - Whether to print out the result of the test and confidence region.

Confidence interval for predictions

```
model.confidence_interval_predicted(x_new, alpha, for_y = False,
                                    print_output = True)
```

Returns (left, right) values of the $(1-\alpha)\%$ confidence intervals for y_{new} or for $m(x_{new})$.

Parameters:

- x_new: *array of shape (1, p)*
  - The new point for which to estimate a confidence interval.
- alpha: *float*
  - Significance level of the confidence interval.
- for_y: *bool, default = False*
  - Whether to compute the interval for y_{new} or for $m(x_{new})$.
- print_output: *bool, default = True*
  - Whether to print out the confidence interval description.

2.2.8 Model Selection

Akaike Information Criterion (AIC)

Given a family of model candidates $\mathcal{A}$ and for some $\alpha \in \mathcal{A}$ we define the Akaike Information Criterion (AIC) as

$$\mathrm{AIC}(\alpha) := n + n \cdot \log(2\pi\hat{\sigma}_\alpha^2) + 2 \cdot \mathrm{card}(\alpha)$$

```
model.compute_aic(x, y)
```

Returns the AIC for a given model.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- y: *array of shape (n, 1)*
  - Vector of response variables.

Bayesian Information Criterion (BIC)

Given a family of model candidates $\mathcal{A}$ and for some $\alpha \in \mathcal{A}$ we define the Bayesian Information Criterion (BIC) as

$$\text{BIC}(\alpha) := n + n \cdot \log(2\pi\hat{\sigma}_\alpha^2) + \log(n) \cdot card(\alpha)$$

```
model.compute_bic(x, y)
```

Returns the BIC for a given model.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- y: *array of shape (n, 1)*
  - Vector of response variables.

3 WLS Model

```
# Import the WLS model
from regression.estimators import WLS
```

The WLS estimator is the solution to the following optimization problem:

$$\hat{\boldsymbol{\beta}}_{\text{wls}} \in \arg\min_{u \in \mathbb{R}^p} \frac{1}{n} \sum_{i \in [n]} w_i (y_i - \mathbf{x}_i^\top \mathbf{u})^2$$

Furthermore, if the design matrix is full-rank then we have that the WLS estimator is unique.

3.1 Initializing the model

The WLS model can be initialized in the same way as the OLS model:

```
wls_model = WLS(standardize, add_bias)
```

and contains the same parameters and attributes.

The WLS class also has the same methods as the OLS class, with the exception of testing methods and a different *fit()* method. It also includes a *feasible_wls()* method. For *predict()*, *score()*, and *mse()* methods, refer to the OLS class for details on their usage.

3.2 Methods

3.2.1 Calculating the WLS estimator

```
wls_model.fit(x, y, w)
```

Calculates the Weighted Least Squares estimator given by $\hat{\boldsymbol{\beta}}_{\text{wls}} = (\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$.

Other values such as $\hat{\mathbf{y}}$ and $\hat{\mathbf{e}}$ are computed in the same way as with OLS.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- y: *array of shape (n, 1)*
  - Vector of response variables.
- w: *array of shape (p, p) or a vector*
  - The desired weights. Can be passed as a diagonal matrix or as a vector.

3.2.2 Hat Matrix

For WLS, we define the Hat Matrix as the following:

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{W}$$

```
wls_model.hat_matrix(x, w)
```

Returns the Hat Matrix under WLS.

3.2.3 Feasible Weight-Least Squares

```
wls_model.feasible_wls(x, y)
```

The Feasible WLS estimator is computed using the following algorithm:

1. Compute the OLS estimator by regressing $\{y_i\}_{i\in[n]}$ on $\{\mathbf{x}_i\}_{i\in[n]}$.
2. Compute the sample residuals $\{\hat{e}_i\}_{i\in[n]}$.
3. Compute a second OLS estimator by regressing $\{\hat{e}_i\}_{i\in[n]}$ on $\{\mathbf{x}_i\}_{i\in[n]}$.
4. Compute $\{\hat{m}(\mathbf{x})_i\}_{i\in[n]}$ using the second OLS estimator and $\{\mathbf{x}_i\}_{i\in[n]}$.
5. Compute the WLS estimator using $\{\hat{m}(\mathbf{x}_i)\}^{-1}_{i\in[n]}$ as weights.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- y: *array of shape (n, 1)*
  - Vector of response variables.

3.2.4 Model Selection with AIC and BIC

We use the same formula (with Hat Matrix defined differently) presented in the OLS section to calculate the AIC and BIC.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- w: *array of shape (n, n)*
  - Matrix of weights.

Akaike Information Criterion (AIC)

```
wls_model.compute_aic(x, y, w)
```

Returns the AIC for a given model.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- y: *array of shape (n, 1)*
  - Vector of response variables.
- w: *array of shape (n, n)*
  - Matrix of weights.

Bayesian Information Criterion (BIC)

```
wls_model.compute_bic(x, y, w)
```

Returns the BIC for a given model.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- y: *array of shape (n, 1)*
  - Vector of response variables.
- w: *array of shape (n, n)*
  - Matrix of weights.

4 Ridge Model

```
# Import the Ridge model
from regression.estimators import Ridge
```

The Ridge estimator is the solution to the following optimization problem:

$$\hat{\boldsymbol{\beta}}_{ridge,\lambda} \in \arg\min_{u \in \mathbb{R}^p} \frac{1}{n} \sum_{i \in [n]} (y_i - \mathbf{x}_i^\top \mathbf{u})^2 + \lambda \|\mathbf{u}\|_2^2$$

which is unique if the design matrix is full rank or $\lambda > 0$.

4.1 Initializing the model

The Ridge model can be initialized in the same way as the OLS and WLS models:

```
ridge_model = Ridge(standardize, add_bias)
```

and contains the same parameters and attributes.

The Ridge class has the same methods as the WLS class, with the exception of *feasible_wls()*. It also has a slightly different *fit()* method. For *predict()*, *score()*, and *mse()* methods, refer to the OLS class for details on their usage.

4.2 Methods

4.2.1 Calculating the Ridge estimator

```
ridge_model.fit(x, y, lambdaa)
```

Calculates the following:

- $\hat{\boldsymbol{\beta}}_{\text{ridge},\lambda} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I_p})^{-1}\mathbf{X}^\top\mathbf{y}$

Other values such as $\hat{\mathbf{y}}$ and $\hat{\mathbf{e}}$ are computed in the same way as with OLS.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- y: *array of shape (n, 1)*
  - Vector of response variables.
- lambdaa: *float*
  - The ridge regularization parameter.

4.2.2 Hat Matrix

For Ridge regression, we define the Hat Matrix as the following:

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I_p})^{-1}\mathbf{X}^\top$$

```
ridge_model.hat_matrix(x, lambdaa)
```

Returns the Hat Matrix under Ridge regression.

4.2.3 Model Selection

We use the same formula (with Hat Matrix defined differently) presented in the OLS section to calculate the AIC and BIC.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- lambdaa: *float*
  - The ridge regularization parameter.

Akaike Information Criterion (AIC)

```
ridge_model.compute_aic(x, y, lambdaa)
```

Returns the AIC for a given model.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- y: *array of shape (n, 1)*
  - Vector of response variables.
- lambdaa: *float*
  - The ridge regularization parameter.

Bayesian Information Criterion (BIC)

```
ridge_model.compute_bic(x, y, lambdaa)
```

Returns the BIC for a given model.

Parameters:

- x: *array of shape (n, p)*
  - Matrix of covariates.
- y: *array of shape (n, 1)*
  - Vector of response variables.
- lambdaa: *float*
  - The ridge regularization parameter.

5 ANOVA Model

```
# Import the model

from regression.anova import ANOVA
```

5.1 Initializing an ANOVA model

```
anova_model = ANOVA(x, y)
```

Parameters

- x: *array of shape (n, 1)*
  - Indicates the group to which each y value belongs.
- y: array of shape (n,1)
  - The values for which to do the analysis.

Attributes

- sorted_y: *array of shape (n, 1)*
  - Contains the sorted values of y, in ascending group order.
- categories: *array of shape (n, 1)*
  - Contains the sorted groups in x, in ascending order, to match with sorted_y.
- G: *int*
  - The number of groups in the data.
- group_counts: *array of shape (G, 1)*
  - Contains the number of observations in each group.

- means: *array of shape (G, 1)*
  - Contains the means of each group.
- bss: *float*
  - The between group sum-of-squares.
- wss: *float*
  - The within group sum-of-squares.
- tss: *float*
  - The total sum-of-squares.

5.2 Methods

5.2.1 Computing the means of each group

```
anova_model.compute_means()
```

This method takes no parameters. It calculates the group means of the data passed in when initializing the ANOVA model. After calling *compute_means*, you can call the *anova_model.means* attribute to access and see the means.

5.2.2 Computing the Sum of Squares

```
anova_model.compute_ss()
```

This method takes no parameters. It calculates the WSS, BSS and TSS of using the data passed in when initializing the ANOVA model. After calling *compute_ss*, you can call

the *anova_model.tss*, *anova_model.wss* and *anova_model.bss* attributes to access and see their values.

The Sum of Squares are calculated like so:

$$\text{TSS} = \text{WSS} + \text{BSS} = \sum_{g\in[G]} \sum_{i\in[n_g]} (y_{g,i} - \bar{y})^2$$

$$\text{WSS} = \sum_{g\in[G]} \sum_{i\in[n_g]} (y_{g,i} - \bar{y}_g)^2$$

$$\text{BSS} = \sum_{g\in[G]} n_g(\bar{y}_g - \bar{y})^2$$

5.2.3 The ANOVA F-Test

```
anova_model.f_test(alpha)
```

Tests the null-hypothesis $H_0 : \beta_0 = \beta_1 = ... = \beta_p$, where β_g is the mean of group $g \in [G]$. Return TRUE if H_0 is accepted, and FALSE if rejected.

Parameters

- alpha: *float*
  - The significance level of the test.

5.2.4 ANOVA Table

```
anova_model.table()
```

This method takes in no parameters and outputs an ANOVA table containing relevant information about the F-test.

6 Utilities

6.1 Categorical Methods

These are methods which are used within some ANOVA methods in the package, but can be used outside of this context, such as data preprocessing in classification tasks.

6.1.1 Creating Dummy Variables for ANOVA

```
from regression.utilities import create_dummy_variables

dummy_matrix = create_dummy_variables(categories)
```

Returns a matrix of shape (n, categories), where each row has value 1 at the in one column and 0 in all others.

Parameters

- categories: *array of shape (n, 1)*
  - The vector specifying the group of each observation.

6.1.2 Sorting the observations by groups for ANOVA

```
from regression.utilities import sort_by_group

sorted_y, sorted_categories = sort_by_group(categories, y)
```

Returns a tuple of sorted observation values and observation groups. The sorting is done in ascending group order (0, 1, 2, ..., G).

Parameters

- categories: *array of shape (n, 1)*
  - The vector specifying the group of each observation.
- y: array of shape (n, 1)
  - The observed values.

7 Statement of Contributions

Mark-Anthony Moisescu:

Hat matrix, AIC and BIC for OLS, WLS and Ridge.

Xiao Qi Li:

Everything else.