MATH 533: Coding Project

raanova: A Regression and ANOVA Python Package

Contents

1 Importing the raanova package

1.1 Requirements

This package requires python $\geq$ 3.12.0, as well as numpy $\geq$ 1.26.0, scipy $\geq$ 1.11.0 and matplotlib $\geq$ 3.8.0. These packages may be installed using

```
$ python -m pip install numpy>=1.26.0 scipy>=1.11.0 matplotlib>=3.8.0
```

1.2 Installation

1.2.1 From PyPI

The package is available on PyPI here, therefore it may be installed using pip with the following command

```
$ python -m pip install raanova
```

This is the recommended approach.

1.2.2 From the repository

It is also possible to install the package directly from the repository using the following command

```
$ python -m pip install git+https://github.com/Scezaquer/MATH-533-
F2023
```

2 How to use raanova to do linear regression

Once the package has been installed, we can start a python script and import all relevant module components

```
>>> from raanova import OLS, WLS, Ridge, Lasso
>>> from raanova.visualisation import display
>>> import numpy as np
```

To perform linear regression on a data set, we first need to get data to use. This may come from anywhere, but in the following sections we will be using a data set generated as follows

```
>>> X = np.random.rand(200, 3) * 100
>>> true_beta = np.atleast_2d([1, 2, 4]).T
>>> epsilon = np.random.randn(200, 1)
>>> Y = X @ true_beta + epsilon
```

where the true model is $y = x_0 + 2x_1 + 4x_2 + \epsilon$ and $\epsilon \sim \mathcal{N}(0, 1)$.

We then pick the estimator we wish to use. The estimators are class objects, so we must create an instance of the class. All available models are listed in the documentation.

```
>>> model = OLS()
```

We may then perform regression using the `fit()` method.

```
>>> beta_hat = model.fit(X, Y)
```

Note that by default, an intercept will be added to the dataset. If you do not wish to have an intercept, you must set the optional argument `intercept = False`.

Now that your model is fitted, you may display all it's informations using `summary()`.

```
>>> model.summary()
Residuals:
Min            Q1       Med        Q3       Max
-2.39962   -0.67489  -0.06838  0.71604  2.95964


Coefficients      Estimates
beta_0            0.08347
beta_1            0.99917
beta_2            1.99813
beta_3            4.00304

Coefficients      Confidence Interval
beta_0            [-0.05887; 0.22582]
beta_1            [0.99674; 1.0016]
beta_2            [1.9957; 2.00056]
beta_3            [4.00063; 4.00545]


R-squared: 0.99999
Naive estimator: 1.02109
Corrected naive estimator: 1.04193

AIC: 3.82575
BIC: 17.01902
```

You may access any individual value using the model's properties. For example

```
>>> model.rsquared
0.9999920455619179
```

The default confidence intervals are 95%. If you wish to have a different value, you can change the `alpha = 0.05` parameter in `fit()`.

In order to make new predictions, you can use the `predict()` method

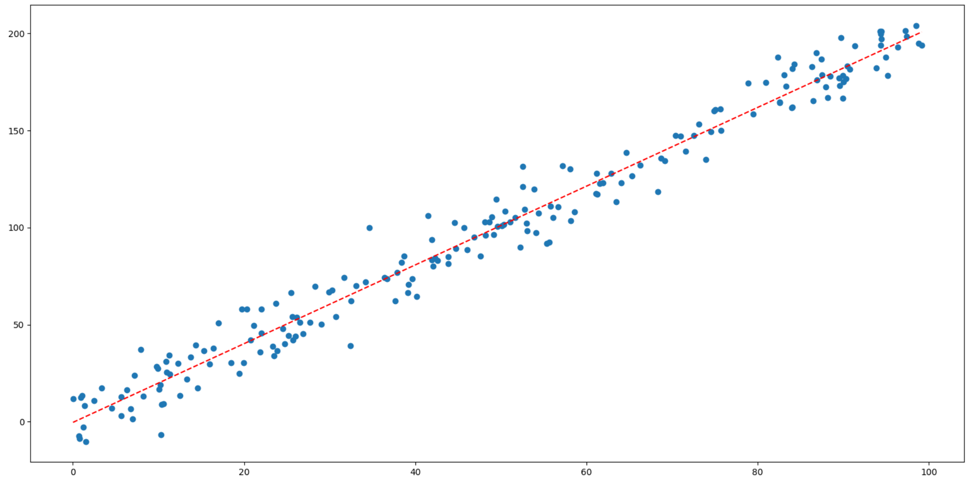
```
>>> X_test = np.random.rand(10, 3) * 100    # Create new unseen data
>>> model.predict(X_test)
array([[378.5578998 ],
       [396.96866941],
       [399.29408612],
       [359.5264308 ],
       [294.3197927 ],
       [184.91960911],
       [355.81027189],
       [266.15079462],
       [348.90667731],
       [322.58849035]])
```

It is important to note that if you set `intercept = False` in the fit method, you will also need to do so in the predict method.

Finally, you may display a graph of your data as well as of your linear fit using the `display()` method. However, this is only available if your design matrix is limited to one covariate.

```
>>> X = np.random.rand(200, 1) * 100
>>> true_beta = np.atleast_2d([2]).T
>>> epsilon = np.random.randn(200, 1) * 10
>>> Y = X @ true_beta + epsilon
>>> model = OLS()
>>> beta_hat = model.fit(X, Y)
>>> display(X, Y, beta_hat) # You may omit beta_hat
```

This will display the following graph.



Other models are made to be used in the exact same way, but complete individual examples are available in the appendix section.

3 Documentation

3.1 OLS

`OLS()`:
Ordinary least squares estimator. Fits a homoscedastic linear model with coefficients $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$ to minimize

$$\mathcal{L}_{\text{OLS}} = ||\mathbf{y} - \mathbf{X}\boldsymbol{\beta}||_2^2$$

OLS methods

`.fit(X,Y, intercept = True, alpha = 0.05)`:
Creates the linear regression model.

`X (NDArray[np.float32])`: $n \times p$ matrix of covariates.

`Y (NDArray[np.float32])`: $n \times 1$ matrix of outcome values.

`intercept (bool)`: Include the intercept in the model. Default value set to `True`.

`alpha (float)`: The alpha used to compute confidence intervals. Default value set to `0.05`.

Returns an array containing the $\hat{\boldsymbol{\beta}}$ values.

`.predict(X)`:
Predicts new values using the input data.

`X (NDArray[np.float32])`: $n \times p$ matrix of covariates.

Returns an array containing the $\hat{\mathbf{y}}$ values.

`.summary()`:
Prints out a summary of the linear regression analysis. Note that the coefficient labels in the table outputs will always start from β_0, regardless of whether the intercept is included or not.

`.conf_interval()`:
Returns the confidence interval with significance level $\alpha = 0.05$.

`.AIC()`:
Returns the AIC.

`.BIC()`:
Returns the BIC.

`.hat()`:
Returns the hat matrix.

`.annihilator()`:
Return the annihilator matrix.

`.residuals()`:
Returns an array containing the sample residuals.

`.rsquared()`:
Returns the value of R^2.

`.sigma_naive()`:
Returns the value of $\hat{\sigma}^2_{\text{naive}}$.

`.sigma_corrected()`:
Returns the value of $\hat{\sigma}^2_{\text{corrected}}$.

3.2 WLS

`WLS()`:
Weighted least squares estimator. Fits a heteroscedastic linear model with coefficients $\boldsymbol{\beta}$ to minimize

$$\mathcal{L}_{\text{WLS}} = ||\mathbf{W}^{1/2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})||_2^2$$

WLS methods

`.fit(X,Y,W, intercept = True)`:
Creates the linear regression model.

`X (NDArray[np.float32])`: $n \times p$ matrix of covariates.

`Y (NDArray[np.float32])`: $n \times 1$ matrix of outcome values.

`W (NDArray[np.float32])`: $n \times n$ weights matrix.

`intercept (bool)`: Include the intercept in the model. Default value set to `True`.

Returns an array containing the $\hat{\boldsymbol{\beta}}$ values.

`.predict(X)`:
Predicts new values using the input data.

`X (NDArray[np.float32])`: $n \times p$ matrix of covariates.

Returns an array containing the $\hat{\mathbf{y}}$ values.

`.summary()`:
Prints out a summary of the linear regression analysis. Note that the coefficient labels in the table outputs will always start from β_0, regardless of whether the intercept is included or not.

`.hat()`:
Returns the hat matrix.

`.annihilator()`:
Return the annihilator matrix.

`.residuals()`:
Returns an array containing the sample residuals.

`.rsquared()`:
Returns the value of R^2.

`.sigma_naive()`:
Returns the value of $\hat{\sigma}^2_{\text{naive}}$.

`.sigma_corrected()`:
Returns the value of $\hat{\sigma}^2_{\text{corrected}}$.

3.3 Ridge

`Ridge()`:
Perform linear least-squares with L_2 regularization. That is we minimize the following loss function with respect to the coefficients $\boldsymbol{\beta}$:

$$\mathcal{L}_{\text{Ridge}} = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2. \tag{1}$$

This particular implementation of the Ridge estimator relies on the data matrix being full rank as the following closed form $\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_p\right)^{-1}\mathbf{X}^T\mathbf{y}$, is used to determine the optimal coefficients.

Ridge methods

`.fit(X, Y, intercept=True, penalty=0.1)`:
Creates the L_2-regularized linear least-squares model.

`X (NDArray[np.float32])`: $n \times p$ matrix of covariates.

`Y (NDArray[np.float32])`: $n \times 1$ matrix of outcome values.

`intercept (bool)`: Include the intercept in the model. Default value set to `True`.

`penalty (float)`: The penalty term applied to the regularization term (λ in (1)). The default value is set to `0.1`.

Returns an array containing the $\hat{\boldsymbol{\beta}}_{\text{Ridge}}$ values.

`.predict(X)`:
Predicts new values using the input data.

`X (NDArray[np.float32])`: $n \times p$ matrix of covariates.

Returns an array containing the $\hat{\mathbf{y}}$ values.

`.summary()`:
Prints out a summary of the linear regression analysis. Note that the coefficient labels in the table outputs will always start from β_0, regardless of whether the intercept is included or not.

`.hat()`:
Returns the hat matrix.

`.annihilator()`:
Return the annihilator matrix.

`.residuals()`:
Returns an array containing the sample residuals.

`.rsquared()`:
Returns the value of R^2.

`.sigma_naive()`:
Returns the value of $\hat{\sigma}^2_{\text{naive}}$.

`.sigma_corrected()`:
Returns the value of $\hat{\sigma}^2_{\text{corrected}}$.

3.4 Lasso

`Lasso()`:
Perform linear least-squares with L_1 regularization. That is we minimize the following loss function with respect to the coefficients $\boldsymbol{\beta}$:

$$\mathcal{L}_{\text{Lasso}} = ||\mathbf{y} - \mathbf{X}\boldsymbol{\beta}||_2^2 + \lambda||\boldsymbol{\beta}||_1^2 \tag{2}$$

This particular implementation of the Lasso estimator uses coordinate descent to find optimal $\hat{\boldsymbol{\beta}}_{\text{Lasso}}$.

Note that selecting the right `step_size` is very important to avoid gradient explosion.

Lasso methods

`.fit(X, Y, intercept=True, penalty=0.1, step_size=0.01, max_iter=1000)`:
Creates the L_1-regularized linear least-squares model.

`X (NDArray[np.float32])`: $n \times p$ matrix of covariates.

`Y (NDArray[np.float32])`: $n \times 1$ matrix of outcome values.

`intercept (bool)`: Include the intercept in the model. Default value set to `True`.

`penalty (float)`: The penalty term applied to the regularization term (λ in (2)). The default value is set to `0.1`.

`step_size (float)`: Step size used when running coordinate descent. Default value is `0.01`.

`max_iter (int)`: Maximum number of iterations before terminating coordinate descent. Default value is `1000`.

Returns an array containing the $\hat{\boldsymbol{\beta}}_{\text{Lasso}}$ values.

`.predict(X)`:
Predicts new values using the input data.

`X (NDArray[np.float32])`: $n \times p$ matrix of covariates.

Returns an array containing the $\hat{\mathbf{y}}$ values.

`.summary()`:
Prints out a summary of the linear regression analysis. Note that the coefficient labels in the table outputs will always start from β_0, regardless of whether the intercept is included or not.

`.residuals()`:
Returns an array containing the sample residuals.

`.rsquared()`:
Returns the value of R^2.

`.sigma_naive()`:
Returns the value of $\hat{\sigma}^2_{\text{naive}}$.

`.sigma_corrected()`:
Returns the value of $\hat{\sigma}^2_{\text{corrected}}$.

3.5 Visualisation

`raanova.visualisation.display(X, Y, betas = None)`:
Displays a scatter plot of the data.

If `betas` is included, the model will be displayed as a line.

Note that this method is only available for data that only includes a single covariate.

`X (NDArray[np.float32])`: $n \times 1$ matrix of covariates.

`Y (NDArray[np.float32])`: $n \times 1$ matrix of outcome values.

`betas (NDArray[np.float32])`: Matrix of coefficients. If this is `None`, the data will be displayed without a model. Default value is `None`.

4 Appendix

4.1 OLS Example

```
# Imports
import numpy as np
from raanova import OLS
from raanova.visualisation import display

# Generate the dataset to perform regression on
# the true model is y = 0*1 + 1*x0 + 2*x1 + 4*x2
X = np.random.rand(200, 3) * 100
true_beta = np.atleast_2d([1, 2, 4]).T
epsilon = np.random.randn(200, 1)
Y = X @ true_beta + epsilon

# Fit
model = OLS()
beta_hat = model.fit(X, Y, intercept=False, alpha=0.05)

# Display model information
model.summary()

# Make predictions on new data
X_test = np.random.rand(10, 3) * 100
y_hat = model.predict(X_test, intercept=False)
print(y_hat)
```

4.2 WLS Example

```
# Imports
import numpy as np
from raanova import WLS

# Generate the dataset to perform regression on
# the true model is y = 0*1 + 1*x0 + 2*x1 + 4*x2
X = np.random.rand(200, 3) * 100
true_beta = np.atleast_2d([1, 2, 4]).T
epsilon = np.random.randn(200, 1)
Y = X @ true_beta + epsilon
W = np.diag(np.full(len(X), 1))

# Fit
model = WLS()
beta_hat = model.fit(X, Y, W, intercept=False)

# Display model information
model.summary()

# Make predictions on new data
X = np.random.rand(10, 3) * 100
y_hat = model.predict(X, intercept=False)
print(y_hat)
```

4.3 Lasso Example

```python
# Imports
import numpy as np
from raanova import Lasso

# Generate the dataset to perform regression on
# the true model is y = 0*1 + 1*x0 + 2*x1 + 4*x2
X = np.random.rand(200, 3) * 100
true_beta = np.atleast_2d([1, 2, 4]).T
epsilon = np.random.randn(200, 1)
Y = X @ true_beta + epsilon

# Fit
model = Lasso()
# Picking the right step size is important to avoid gradient
    explosion
beta_hat = model.fit(
    X, Y, intercept=True, penalty=0.1, step_size=0.0001, max_iter
    =1000)

# Display model information
model.summary()

# Make predictions on new data
X = np.random.rand(10, 3) * 100
y_hat = model.predict(X, intercept=True)
print(y_hat)
```

4.4 Ridge Example

```python
# Imports
import numpy as np
from raanova import Ridge

# Generate the dataset to perform regression on
# the true model is y = 0*1 + 1*x0 + 2*x1 + 4*x2
X = np.random.rand(200, 3) * 100
true_beta = np.atleast_2d([1, 2, 4]).T
epsilon = np.random.randn(200, 1)
Y = X @ true_beta + epsilon

# Fit
model = Ridge()
beta_hat = model.fit(X, Y, intercept=False)

# Display model information
model.summary()

# Make predictions on new data
X = np.random.rand(10, 3) * 100
y_hat = model.predict(X, intercept=False)
print(y_hat)
```